

\$25.00

A **PAPERBYTE**™ BOOK

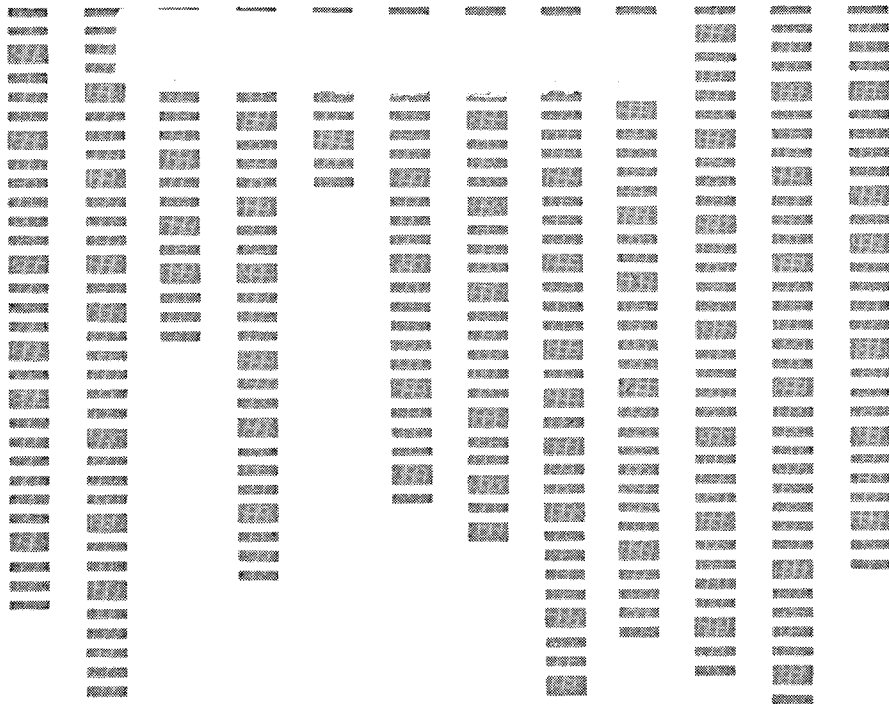


RA6800ML

AN M6800 RELOCATABLE MACRO ASSEMBLER
by Jack E. Hemenway

RA6800ML

AN M6800 RELOCATABLE MACRO ASSEMBLER by Jack E. Hemenway



BYTE Publications, Inc.
70 Main Street
Peterborough, New Hampshire 03458

The author of the programs provided with this book has carefully reviewed them to ensure their performance in accordance with the specifications described in the book. The author, however, makes no warranties whatever concerning the programs, and assumes no responsibility or liability of any kind for errors in the programs or for the consequences of any such errors. The programs are the sole property of the author and have been registered with the United States Copyright Office.

Copyright © 1978 BYTE Publications Inc. All Rights Reserved. BYTE and PAPERBYTE are Trademarks of BYTE Publications Inc. No part of this book may be translated or reproduced in any form without the prior written consent BYTE Publications Inc.

Program Copyright © 1977 by Jack E. Hemenway, Boston, MA. All Rights Reserved.

Library of Congress Cataloging in Publication Data

Hemenway, Jack E

RA6800ML resident macro linking assembler for the Motorola M6800.

1. Motorola 6800 (Computer) — Programming. 2. RA6800ML (Computer program) 3. Assembling (Electronic computer) I. Grapple, Robert D., joint author. II. Title.

QA76.8.M67H45

001.6'425

78-22084

ISBN 0-931718-10-4

Printed in the United States of America.

Table of Contents

TO BEGIN WITH	v
THE SOURCE LANGUAGE	1
Instruction Format	1
Addressing Modes	2
Address Type Formats	3
Pseudo Instructions	5
Macros	7
Program Linkage	7
THE ASSEMBLER	11
Assembler Modules Overview	11
Tables	12
Stacks	13
Utility Routines	13
Listings Routines	14
Mnemonic and Symbol Table Routines	16
Input and Output Routines	19
Pseudo Instruction Processing	24
Opcode Processing Routines	39
Address Processing	48
Lexical Analysis Routines	52
Evaluation Routine	56
INTERFACING AND USING THE ASSEMBLER	59
IO Interface Conventions	59
Tape Driver Routines	59
Disk Driver Routines	60
Assembler Loading and Execution	60
Loading the Object Code	61
Source Tape Format	61
Object Tape Format	61
APPENDICES	63
Appendix A: Error Messages	65
Appendix B: Capacities	65
Appendix C: Notes from a User: Implementation of RA6800ML	67
Appendix D: RA6800ML Assembly Language Object Code in Absolute Hexadecimal Format	69
Appendix E: PAPERBYTE™ Bar Code Representation of RA6800ML in Absolute Format	79
Appendix F: Input and Output Routines for RA6800ML in Absolute Format with PAPERBYTE™ Bar Code Representation	89
Appendix G: Assembly Language Source Listing of RA6800ML	95
Appendix H: ASCII Text Listing of the Relocatable Format Object Code for RA6800ML	139
Appendix I: PAPERBYTE™ Bar Code Representation of Relocatable Format Object Code for RA6800ML	145
Appendix J: Cassette Tape IO Listing	163
Appendix K: ICOM Floppy Disk IO Listing	169
INDEX	171

To Begin With . . .

RA6800ML, a resident Macro Assembler for the Motorola 6800 Microprocessor, is a two pass assembler designed to run on a minimum system of 16 K bytes of memory, a system console such as a Teletype, a system monitor such as the Motorola MIKBUG read only memory program or the ICOM Floppy Disk Operating System (FDOS), and some form of mass file storage such as dual cassette recorders or a floppy disk. A system monitor other than those mentioned above could be used by simply changing two IO jumps in the Assembler, a jump to the input-a-character routine INEEE and a jump to the output-a-character routine OUTEEE, and by supplying functionally equivalent IO routines for the user's specific system.

The Assembler can produce a program listing, a sorted Symbol Table listing, and relocatable object code. The object code is loaded and linked with other assembled modules using the Linking Loader LINK6800. The companion PAPERBYTE™ publication *LINK68 — Linking Loader for Motorola 6800* gives details on how to use the Linking Loader.

This book is divided into four major sections. In the section THE SOURCE LANGUAGE, a detailed description of the 6800 assembly language and its components is given. The instruction and address type formats are outlined in addition to details about the pseudo instructions and macro facilities. This section provides the necessary background for coding programs in the 6800's assembly language and understanding the operations of the Assembler.

The section on THE ASSEMBLER describes the actual routines which make up the Assembler. Each subsection presents a logical collection of routines which provide a particular function. In addition to short descriptions of the routines, a cross reference is given showing all calling and called by routines. Additional information about pointers, flags, and temporary variables is supplied. Finally, detailed flowcharts of each routine are provided.

The exact IO interface needed for using the Assembler naturally depends on the actual configuration of the user's system. In INTERFACING AND USING THE ASSEMBLER sample IO routines for a tape system and floppy disk system are examined. Tips are given on how to design IO routines (or modify those provided as examples) to fit the user's system. Finally, information on loading and executing the Assembler, as well as source and object tape formats, are provided.

Section five is the appendices which contain error messages generated by the Assembler, the Assembler and same IO driver source code listings, the bar code representation of the assembler's relocatable object file, an implementation guide for bootstrapping RA6800ML without the use of the linking loader LINK68, and the Assembler and IO routines in absolute formats for the bootstrap process.

Finally, a detailed INDEX is included for quick reference to a variety of items.

In this book is what I believe to be a complete set of documentation for the 6800 assembler program. Every flowchart, every listing, every item was included for one purpose: to provide the user with everything needed for the use of modification of the Macro Assembler.

In addition, it was my express purpose to provide everything necessary so that the user can easily learn what he or she needs to know about the system. By providing not only the 6800 assembler language description, but also a source code listing and detailed description of every routine of the Assembler, I intend to provide the user with an opportunity to learn about the nature of assembler design and implementation as well as simply acquiring a useful software tool. It is through this kind of encouragement that I hope to advance the state of the art of home computing.

Jack E. Hemenway

The Source Language

Instruction Format

A source language statement consists of a label, an operation code, an operand, and comments. The label is used when needed as a reference point for other statements. The operation code may be a mnemonic machine operation, a pseudo instruction, or a Macro call (a reference to the Macro's name). An operand may be an expression consisting of an alphanumeric symbol, a number, a special character, or any of these combined with arithmetic operators; or in certain instances there may be no operand at all. The comments are entirely optional. The fields in a source statement are separated by at least one space character (20 hexadecimal). This source language definition is based on the original Motorola 6800 assembly language, with minor omissions and major extensions such as the Macro facility.

Statement Characteristics

The fields of the source statement appear in the following order:

[label] opcode operand(s) [comments]

The items in brackets ([]) are optional.

Field Delimiters

One or more spaces separate the fields of a statement. An End-of-Statement mark (carriage return) terminates the entire statement. A single space following an End-of-Statement mark from the previous statement indicates the absence of the label field.

Character Set

The ASCII characters recognized by the Assembler are as follows:

A through Z } "alphabetic" } "alphanumeric"
0 through 9 } "numeric" }
* (asterisk)
+ (plus)
- (minus)

/ (slash)
\$ (dollar sign)
' (apostrophe, single quote mark)
, (comma)
(pound sign)
& (ampersand)
(space)

Any other valid ASCII characters may appear in the comments field.

The letters A through Z, and the numbers 0 through 9 may be used in an alphanumeric symbol. In the first position of the label field an asterisk indicates a comment line; in the operand field it represents the value of the program location counter for the current instruction if it is in the first position; otherwise it is recognized as the multiplication operator for an expression.

The plus, minus, slash, and asterisk are used as operators in arithmetic expressions.

The pound sign is used to indicate the immediate addressing mode, the dollar to indicate hexadecimal numbers, the apostrophe to indicate ASCII strings, the ampersand to indicate substitutable parameters in Macro definitions, and the comma to separate operands.

Spaces separate fields of a statement and may also be used to format the output listing.

Statement Length

A statement may be up to 72 characters long.

Label Field

The *label* field serves to identify the statement and may be used as a reference by other statements in the program.

This field starts immediately following an End-of-Statement mark and is terminated by a space. A space in position one of a statement indicates that the statement is unlabeled.

Label Symbol

A label is composed of from one to six characters. The first character must be an alphabetic character. The remain-

ing characters must be alphanumeric characters. If the label is composed of more than six characters, the Assembler truncates the symbol to six characters.

An asterisk in position one indicates that the entire statement is a comment. An asterisk in any position of the *label* field other than the first position is illegal.

Opcode Field

The operation code (opcode) defines an operation to be performed by the computer or by the Assembler. This field may contain an operation code, a pseudo instruction, or a Macro reference. The *opcode* field follows the *label* field and is separated from it by at least one space. If there is no label, this field may begin anywhere after position one. The *opcode* field is terminated by a space.

Operand Field

The meaning and format of the *operand* field is dependent on the type of operation code used in the source statement.

This field follows the *opcode* field and is separated from it by at least one space. When dual operand instructions are used, the first operand must be either an "A" or a "B", indicating the A or B accumulator, respectively. The single characters "A" or "B" must be preceded and followed by one or more spaces. The *operand* field is terminated by a space except when there are no comments; in that case it may be terminated by an End-of-Statement mark.

Symbolic Terms

A symbolic term (symbol) follows the same rules for the formation of labels. A symbol used in the *operand* field must be defined elsewhere in the program. An asterisk may be used to refer to the value of the location counter at the time the source statement is encountered.

Numeric Terms

A numeric term (number) may be either decimal or hexadecimal. A decimal number is represented by one to five decimal digits within the range 0 to 65535. A hexadecimal number is indicated by one to four hexadecimal digits within the range 0 to FFFF and is preceded by a dollar sign (\$).

Strings

An ASCII string is any sequence of valid ASCII characters preceded by a single quote mark and followed by a single quote mark. If an embedded apostrophe is needed, two apostrophes are used (which count as a single character.) The value of a string is formed by the 8 bit ASCII characters enclosed between the delimiting apostrophes.

Expression Operators

The asterisk, symbols, and numbers may be joined by the four arithmetic operators (+ - * /) to form arithmetic expressions. The Assembler evaluates expressions from left to right *without* regard to precedence or operator hierarchy. A fractional result, if obtained *during* the evaluation of an

expression, is truncated to an integer value.

Example:

$$3/2 + 1 = 1 + 1 = 2$$

Macro Call Argument Lists

Macros are passed arguments by placing the arguments in the *operand* field separated by commas. The actual arguments are substituted as character strings into the positions of the corresponding dummy arguments in the macro definition. If comments are to be included in the statement, a comma must follow the last argument.

Evaluation of Symbols and Expressions

Because of the two pass nature of the Assembler, only one level of forward referencing is legal in the use of symbols and expressions in the *operand* field of source statements.

Comment Field

A *comment* field may be included in a source statement as long as it is separated by at least one space from the *operand* field. However, when a comment is included on a macro call statement, the last macro argument must be followed by a comma.

Addressing Modes

Dual Operand

These instructions require two operands in the *operand* field. The first operand must always reference either the A or B accumulator and is separated from the second operand by at least one space. The second operand is formed in accordance with the rules for Direct, Extended, Immediate, or Indexed addressing.

Accumulator

These instructions reference only one operand in the *operand* field; this operand is always either the A or B accumulator represented by the single character "A" or "B".

Inherent

These instructions require no operands, as the information needed is implied by the instruction itself.

Indexed

These instructions reference the Index register X. The *operand* field of the instruction is evaluated and placed in the second byte of the instruction. When the instruction is executed the contents of this byte are added to the Index register to form the complete address. The format is:

$$\left\{ \begin{array}{l} \text{number} \\ \text{symbol} \\ \text{expression} \end{array} \right\}, X$$

where number, symbol, or expression evaluates to a value between 0 and 255. If a larger value is generated only the

low order eight bits are used.

Examples:

5, X
TEST, X
G + 55, X

Immediate

For these instructions the actual value of the operand is placed in the object code itself following the instruction machine code. The Immediate mode of addressing is indicated by preceding the operand with a pound sign (#). The format is:

{ number
symbol
expression
ASCII string }

Examples:

#100
#TEST
#ABC

Relative

These two byte instructions are always branch instructions. The branch address is taken relative to the current contents of the Program Counter. The second byte contains a signed number in two's complement notation that specifies the relative branch address. The address of the destination of the branch must be in the range of -126 to +129 relative to the address of the first byte of the branch instruction. The format is:

{ number
symbol
expression }

Direct and Extended

For those instructions that allow it, the Assembler will select the Direct mode of addressing if the evaluated operand address is in the numerical range of 0 to 255, provided that the evaluated address is not relocatable or common. In these cases the Extended mode will be selected. The Direct mode generates two bytes of machine code: the second byte contains the eight bit address in unsigned binary; the upper 8 bits of the sixteen bit address are assumed to be zeroes. If the evaluated operand is greater than 255 (ie: Extended mode is used) then the Assembler generates three bytes of machine code. The second and third bytes contain the sixteen bit unsigned binary address. The source language format is:

{ number
symbol
expression }

Address Type Formats

There are nine basic address type formats which employ Immediate, Direct, Extended, Relative, Indexed, Accumu-

lator (ACCX), or Inherent modes of addressing. In the formats given below, the bracket symbols { } indicate that any one of the enclosed items may be chosen for the position indicated (but only one). In addition, the symbol $\emptyset$ will be used to indicate that one or more blanks must appear in that position. All 6800 instructions which use the format are shown at the left in these examples.

Address Type 1

Immediate mode (two bytes):

{ ADC ADD
AND BIT
CMP LDA
ORA SBC
SUB } $\emptyset$ { A
B } $\emptyset$ # { number
symbol
expression
ASCII string }

Direct mode (two bytes) or Extended mode (three bytes):

{ ADC ADD AND
BIT CMP LDA
ORA SBC SUB } $\emptyset$ { A
B } $\emptyset$ { number
symbol
expression }

Indexed mode (two bytes):

{ ADC ADD AND
BIT CMP LDA
ORA SBC SUB } $\emptyset$ { A
B } $\emptyset$ { number
symbol
expression },X

Address Type 2

Extended mode (three bytes) or Direct mode (two bytes):

STA $\emptyset$ { A
B } $\emptyset$ { number
symbol
expression }

Indexed mode (two bytes):

STA $\emptyset$ { A
B } $\emptyset$ { number
symbol
expression },X

Address Type 3

Accumulator mode (one byte):

$$\left\{ \begin{array}{l} \text{ASL} \quad \text{ASR} \quad \text{CLR} \quad \text{COM} \\ \text{DEC} \quad \text{INC} \quad \text{LSR} \quad \text{NEG} \\ \text{ROL} \quad \text{ROR} \quad \text{TST} \end{array} \right\} \neq \left\{ \begin{array}{l} \text{A} \\ \text{B} \end{array} \right\}$$

Indexed mode (two bytes):

$$\left\{ \begin{array}{l} \text{CPX} \\ \text{LDS} \\ \text{LDX} \end{array} \right\} \neq \left\{ \begin{array}{l} \text{number} \\ \text{symbol} \\ \text{expression} \end{array} \right\}, X$$

Extended mode (three bytes):

$$\left\{ \begin{array}{l} \text{ASL} \quad \text{ASR} \quad \text{CLR} \quad \text{COM} \\ \text{DEC} \quad \text{INC} \quad \text{LSR} \quad \text{NEG} \\ \text{ROL} \quad \text{ROR} \quad \text{TST} \end{array} \right\} \neq \left\{ \begin{array}{l} \text{number} \\ \text{symbol} \\ \text{expression} \end{array} \right\}$$

Address Type 6

Direct mode (two bytes) or Extended mode (three bytes):

$$\left\{ \begin{array}{l} \text{STX} \\ \text{STS} \end{array} \right\} \neq \left\{ \begin{array}{l} \text{number} \\ \text{symbol} \\ \text{expression} \end{array} \right\}$$

Indexed mode (two bytes):

$$\left\{ \begin{array}{l} \text{ASL} \quad \text{ASR} \quad \text{CLR} \quad \text{COM} \\ \text{DEC} \quad \text{INC} \quad \text{LSR} \quad \text{NEG} \\ \text{ROL} \quad \text{ROR} \quad \text{TST} \end{array} \right\} \neq \left\{ \begin{array}{l} \text{number} \\ \text{symbol} \\ \text{expression} \end{array} \right\}, X$$

Indexed (two bytes):

$$\left\{ \begin{array}{l} \text{STX} \\ \text{STS} \end{array} \right\} \neq \left\{ \begin{array}{l} \text{number} \\ \text{symbol} \\ \text{expression} \end{array} \right\}, X$$

Address Type 4

Accumulator mode (one byte):

$$\left\{ \begin{array}{l} \text{PSH} \\ \text{PUL} \end{array} \right\} \neq \left\{ \begin{array}{l} \text{A} \\ \text{B} \end{array} \right\}$$

Address Type 7

Extended mode (three bytes):

$$\left\{ \begin{array}{l} \text{JMP} \\ \text{JSR} \end{array} \right\} \neq \left\{ \begin{array}{l} \text{number} \\ \text{symbol} \\ \text{expression} \end{array} \right\}$$

Address Type 5

Immediate mode (three bytes):

$$\left\{ \begin{array}{l} \text{CPX} \\ \text{LDS} \\ \text{LDX} \end{array} \right\} \neq \# \left\{ \begin{array}{l} \text{number} \\ \text{symbol} \\ \text{expression} \\ \text{ASCII string} \end{array} \right\}$$

Indexed mode (two bytes):

$$\left\{ \begin{array}{l} \text{JMP} \\ \text{JSR} \end{array} \right\} \neq \left\{ \begin{array}{l} \text{number} \\ \text{symbol} \\ \text{expression} \end{array} \right\}, X$$

Direct mode (two bytes) or Extended mode (three bytes):

$$\left\{ \begin{array}{l} \text{CPX} \\ \text{LDS} \\ \text{LDX} \end{array} \right\} \neq \left\{ \begin{array}{l} \text{number} \\ \text{symbol} \\ \text{expression} \end{array} \right\}$$

Address Type 8

Relative mode (two bytes):

$$\left\{ \begin{array}{l} \text{BCC} \quad \text{BCS} \quad \text{BEQ} \quad \text{BGE} \quad \text{BGT} \\ \text{BHI} \quad \text{BLE} \quad \text{BLT} \quad \text{BMI} \quad \text{BRA} \\ \text{BSR} \quad \text{BVC} \quad \text{BVS} \end{array} \right\} \neq \left\{ \begin{array}{l} \text{number} \\ \text{symbol} \\ \text{expression} \end{array} \right\}$$

Address Type 9

Inherent mode (one byte):

{	ABA	CBA	CLC	CLI	CLV
	DAA	DES	DEX	INS	INX
	NOP	RTI	RTS	SBA	SEC
	SEI	SEV	SWI	TAB	TAP
	TBA	TPA	TSX	TXS	WAI

Pseudo Instructions

In this section, the set of “pseudo instructions” which are defined for this Assembler are described. A pseudo instruction (sometimes called a pseudo operation or “pseudop”) gives instructions to the Assembler itself, not to the machine. Often a pseudo instruction results in no object code generation. Sometimes data is allocated with or without initial values. Pseudo operations are also differentiated from Macro instructions in that a Macro is user defined while the pseudo operation is defined by the Assembler.

CMN

This is used to reserve (declare) an area in Common for interprogram data communication. The syntax is:

⌀ CMN ⌀ symbol, { number
symbol
expression }

The second operand is evaluated and the value obtained is used to reserve that amount of bytes in the Common area. The CMN must not be labeled.

END

This terminates a program. It marks the physical end of the source language program. The last statement of a program must be an END statement. The END statement must not be written with a *label*, generates no object code, and has no operand.

ENT

This is used to declare an “entry point”, a symbol that may be referenced by separately assembled programs. The syntax is:

⌀ ENT ⌀ symbol

This statement must not be labeled and the *operand* field must contain a symbol that is defined elsewhere in the program.

EXT

This is used to declare an “external reference”, a symbol which may be referenced by the program but which is defined in some other program. The syntax is:

⌀ EXT ⌀ symbol

This statement must not be labeled. The symbol in the *operand* field must be declared by an ENT statement in the program in which it is defined.

EQU

This assigns to a symbol a value other than the value normally assigned by the Program Location Counter. The syntax is:

label ⌀ EQU ⌀ { number
symbol
expression }

The EQU statement must be labeled. Symbols appearing in the *operand* field must be previously defined in the source program. This pseudo instruction generates no object code.

FCB

This generates one byte of object code. An eight bit unsigned binary number corresponding to the value of the operand is stored in the object code. The format is:

[label] ⌀ FCB ⌀ { number
symbol
expression }

If the operand evaluates to a value larger than 255, only the least significant eight bits will be used. The FCB pseudo instruction may be labeled.

FCC

This translates strings of characters into their seven bit ASCII code. The format is:

[label] ⌀ FCC ⌀ ASCII string

The ASCII string is text enclosed between apostrophes. If an apostrophe is needed in the text it is represented by two apostrophes; however, only one will be put into the object code. This statement may be labeled.

FDB

This generates two bytes of object code. A sixteen bit unsigned binary number corresponding to the value of the operand is stored in the object code. The format is:

[label] ⌀ FDB ⌀ { number
symbol
expression }

This statement may be labeled.

IF

This is used to cause the Assembler to process the following code normally if the value of the operand is not zero; but the Assembler is to ignore all source statements until a matching NIF statement is encountered if the value of the operand is zero. The format is:

⌀ IF ⌀ { number
symbol
expression }

The IF pseudo instruction must not be labeled but must have an operand. This implements the simplest form of conditional assembly and can be used either within or independent of a Macro.

MAC

This is used in the definition of a Macro. All statements following the MAC instruction up to the next MEND are stored in the Macro Table as a Macro definition. The syntax is:

label ⌀ MAC [C]

A label is required on the MAC statement. The label is the symbol (its name) by which a Macro is expanded or called. The *operand* field may contain a "C"; the "C" is used to specify whether or not comment lines in the Macro definition are to be stored in the Macro Table. A "C" in the *operand* field indicates that all comment lines are to be included in the expansion of the Macro. By omitting the "C", the user can lower the main memory requirements needed to store the Macro definition. Macro definitions may not be nested but may contain calls to other Macros.

MEND

This indicates the end of a Macro definition. It must not have a label or an operand.

NAM

This names the program. The syntax is:

⌀ NAM ⌀ symbol

The symbol in the *operand* field is passed to the Linking Loader as an Entry point. It must not be used as a label elsewhere in the program. A NAM pseudo instruction must be included in each program as the first statement.

NIF

This is used as a terminator to an IF pseudo operation. It must be unlabeled and has no operand.

PAG

This causes the listing device to advance to the top of the next page. This statement does not appear on the listing, causes no object code to be generated, and must be unlabeled.

RMB

This reserves a block of memory whose length is the value of the operand. The syntax is:

[label] ⌀ RMB ⌀ { number
symbol
expression }

This statement may be labeled. The block of memory reserved is cleared to zeroes. Symbols used in the *operand* field must have been previously defined in the program.

Example 1: A Macro Prototype

```

line 1  LOOP MAC C
line 2      LDX #&0    LOAD X WITH ARGUMENT 0
line 3      LDA B #&1  LOAD B WITH ARGUMENT 1
line 4  *
line 5      DEX        X:=X-1
line 6      BNE *-1    X.ne. 0
line 7  *
line 8      DEC B      B:=B-1
line 9      BNE *-5    B.ne. 0
line 10 *
line 11     RTS        ALL DONE
line 12     MEND

```

Line 1 is the header. It names the Macro as LOOP and specifies that comment lines in the body are to be stored with the Macro definition in the Macro Table.

Lines 2 and 3 are source statements with substitutable arguments (parameters) in the variable field (&0, &1). A parameter is recognized by the presence of the ampersand. The digit after the "&" is the argument number. Ten arguments is the maximum number of arguments allowable in a single Macro, numbered 0 thru 9.

Lines 4 thru 11 make up the rest of the prototype body.

Line 12 is the termination line.

Lines 2 thru 11 are stored in the Macro Table by the Assembler for later use. If the "C" had not been on the header statement, lines 4, 7 and 10 would not have been saved.

The Macro name "LOOP" is stored in the Symbol Table with a pointer to the location of the Macro definition in the Macro Table.

A typical reference to the Macro "LOOP" might be:

LABEL LOOP 1000, 12

This would expand into the following:

```

          LDX #1000  LOAD X WITH ARGUMENT 0
          LDA B #12  LOAD B WITH ARGUMENT 1
*
          DEX        X:=X-1
          BNE *-1    X.ne. 0
*
          DEC B      B:=B-1
          BNE *-5    B.ne. 0
*
          RTS        ALL DONE

```

The argument "1000" is substituted for &0, and the argument "12" is substituted for &1.

Example 1: An example of a Macro prototype and a reference to the Macro.

Macros

Macros are sections of code which are defined once at the beginning of a program and used and referenced by a mnemonic code (with or without parameters) in the rest of the program.

Usually the code one places in a macro consists of statements that are repeated many times at different places throughout a program. Macros provide a shorthand notation for repeating these sections of code.

The statements of any given Macro are grouped in one place at the beginning of the program. This "Macro definition" is preceded by the MAC pseudo instruction and followed by a series of instructions, and finally the MEND pseudo instruction. The Macro is named by placing the name in the *label* field of the MAC statement. The Macro is called by placing the name of the Macro in the *opcode* field of a statement, and any parameters to be passed to the Macro are placed in the *operand* field separated by commas.

The expansion of a Macro is sometimes thought of as an open subroutine in that it produces the same inline code every time. The inline code is inserted in the normal flow of the program so that the generated statements are assembled with the rest of the program. *[Unlike a conventional subroutine, the open subroutine is repeated every time it is used without a call and return linkage . . . Carl Helmers.]*

The Macro definition is also known as the prototype. The source statements included in the prototype may be any legal Assembler or processor instruction except for another MAC pseudo operation.

Macro prototypes are of the form:

<i>header</i>	<i>label</i>	MAC [C]	
<i>body</i>			} any statements
			
<i>termination</i>			
		MEND	

Where:

label is the name of the Macro;

"C" is an optional operand to control the storing of comment lines along with the prototype body;

body is the sequence of source statements;

termination is the line containing the pseudo instruction MEND.

MEND is recognized by the Assembler as the end of the Macro definition.

Program Linkage

Linking pseudo instructions are used to provide a means of communications between a main program and its subroutines, or among several subprograms that are to be linked together to run as a single program.

Common

Ø CMN Ø symbol, { number
symbol
expression }

CMN reserves a block of storage locations that may be used in common by several programs. Each symbol (the first operand) identifies a segment of the block for the subprogram in which the CMN statement appears. The second operand is the length of the related segment.

Any number of CMN statements may appear in a subprogram. Storage locations in Common are assigned contiguously. The length of the Common block is equal to the sum of the lengths of all segments named in CMN statements in the subprogram.

To refer to the Common block, other subprograms must also include a CMN statement. The segment names and lengths may be the same or they may differ. Regardless of the names and lengths specified in the separate subprograms, there is only one Common block for the combined set of programs. It has the same relative origin; the content of the *n*th byte of Common storage is the same for all subprograms. Thus a key part of designing a large user software system is allocation and definition of the common variables used to communicate between separate modules.

The segment names that appear in the CMN statements can be used in the *operand* fields of EQU, FDB, or any memory reference statement; they may not be used as labels elsewhere in the program. All references to Common are relocatable.

The user establishes the origin of the Common block when the Linking Loader is executed.

Note that two or more subprograms may declare Common blocks that differ in size, although example 2 shows the same size for both programs.

Entry

Ø ENT Ø symbol

ENT defines entry points to the program or subprogram. Symbol is an assigned label for some statement in the program. Entry points allow another program to refer to the program in which the ENT occurs. All entry points must be defined in the program.

External

Ø EXT Ø symbol

EXT designates labels in other programs that are referenced in this program. Symbol must be defined by an ENT in some other program.

The CMN pseudo operation is provided to allow data communication and the EXT and ENT pseudo instructions are provided to allow control communication between separately assembled or compiled subprograms that are linked together to form a single program to be executed as a unit.

The following Macro prototypes may be useful to the programmer. Each was designed with a special purpose in mind, such as an arithmetic operation on a 16 bit integer quantity.

To increment a 16 bit quantity, the following Macro could be used:

```
INC16      MAC
           INC  &0+1
           BNE  *+5
           INC  &0
           MEND
```

Here there is only one parameter, &0. When the Macro is referenced, a parameter would be included as in the statement:

```
INC16 AAA
```

This would generate the instructions needed to increment variable AAA by one:

```
INC  AAA+1
BNE  *+5
INC  AAA
```

Similar Macro prototypes, a sample reference, and the code generated by that reference are given below. Note that in these examples, the macros PSHX, PULX, PSHREG and PULREG use self-modifying code techniques and *will not work* if a program using them is stored in read only memory.

To decrement a 16 bit quantity:

Prototype:		Sample reference followed by generated code:
DEC16	MAC	DEC16 BBB
	TST &0+1	TST BBB+1
	BNE *+5	BNE *+5
	DEC &0	DEC BBB
	DEC &0+1	DEC BBB+1
	MEND	

To add two 16 bit quantities together, placing the sum into a third 16 bit variable (note that 3 parameters are needed):

Prototype:		Sample reference followed by generated code:
ADD16	MAC	ADD16 AAA,BBB,CCC
	LDA A &0+1	LDA A AAA+1
	LDA B &0	LDA B AAA
	ADD A &1+1	ADD A BBB+1
	ADC B &1	ADC B BBB
	STA A &2+1	STA A CCC+1
	STA A &2	STA A CCC
	MEND	

To subtract two 16 bit quantities, placing the difference into the first 16 bit variable:

Prototype:		Sample reference followed by generated code:
SUB16	MAC	SUB16 AAA,BBB,AAA
	LDA A &0+1	LDA A AAA+1
	LDA B &0	LDA B AAA
	SUB A &1+1	SUB A BBB+1
	SBC B &1	SBC B BBB
	STA A &2+1	STA A AAA+1
	STA B &2	STA B AAA
	MEND	

To push the X register onto a stack:

Prototype:		Sample reference followed by generated code:
PSHX	MAC	PSHX
	DES	DES
	DES	DES
	STS *+4	STS *+4
	STX *+1	STX *+1
	MEND	

To pull the X register off of the stack:

Prototype:		Sample reference followed by generated code:
PULX	MAC	PULX
	STS *+4	STS *+4
	LDX *+1	LDX *+1
	INS	INS
	INS	INS
	MEND	

To push the X, A, and B registers onto a stack (note the nested Macro reference):

Prototype:		Sample reference followed by generated code:
PSHREG	MAC	PSHREG
	PSHX	PSHX
	PSH A	DES
	PSH B	DES
	MEND	STS *+4
		STX *+1
		FSH A
		PSH B

To pull the B, A, and X registers off of the stack (again, note the nested Macro reference):

Prototype:		Sample reference followed by generated code:
PULREG	MAC	PULREG
	PUL B	PUL B
	PUL A	PUL A
	PULX	PULX
	MEND	STS *+4
		LDX *+1
		INS
		INS

Example 2: This example shows the linkage of two external routines via the Common Block. The Common Block layout shows the locations of the symbolic terms defined in PROG1 and PROG2 within the block. Note that the LDA A instruction in both routines refer to the same COMMON block byte.

Example 2: A Common Block and Its References				
NAM	PROG1			
CMN	AAA,5	ALLOCATE 5 BYTES OF COMMON		
CMN	BBB,10	ALLOCATE 10 BYTES OF COMMON		
CMN	CCC,10	ALLOCATE 10 BYTES OF COMMON		
.				
.				
LDA	A	BBB+1	LOAD BYTE 2 OF SEGMENT BBB	
.				
END				
NAM	PROG2			
CMN	DDD,2	ALLOCATE 2 BYTES OF COMMON		
CMN	EEE,2	ALLOCATE 2 BYTES OF COMMON		
CMN	FFF,1	ALLOCATE 1 BYTE OF COMMON		
CMN	GGG,20	ALLOCATE 20 BYTES OF COMMON		
.				
LDA	A	GGG+1	LOAD BYTE 2 OF SEGMENT GGG	
.				
END				

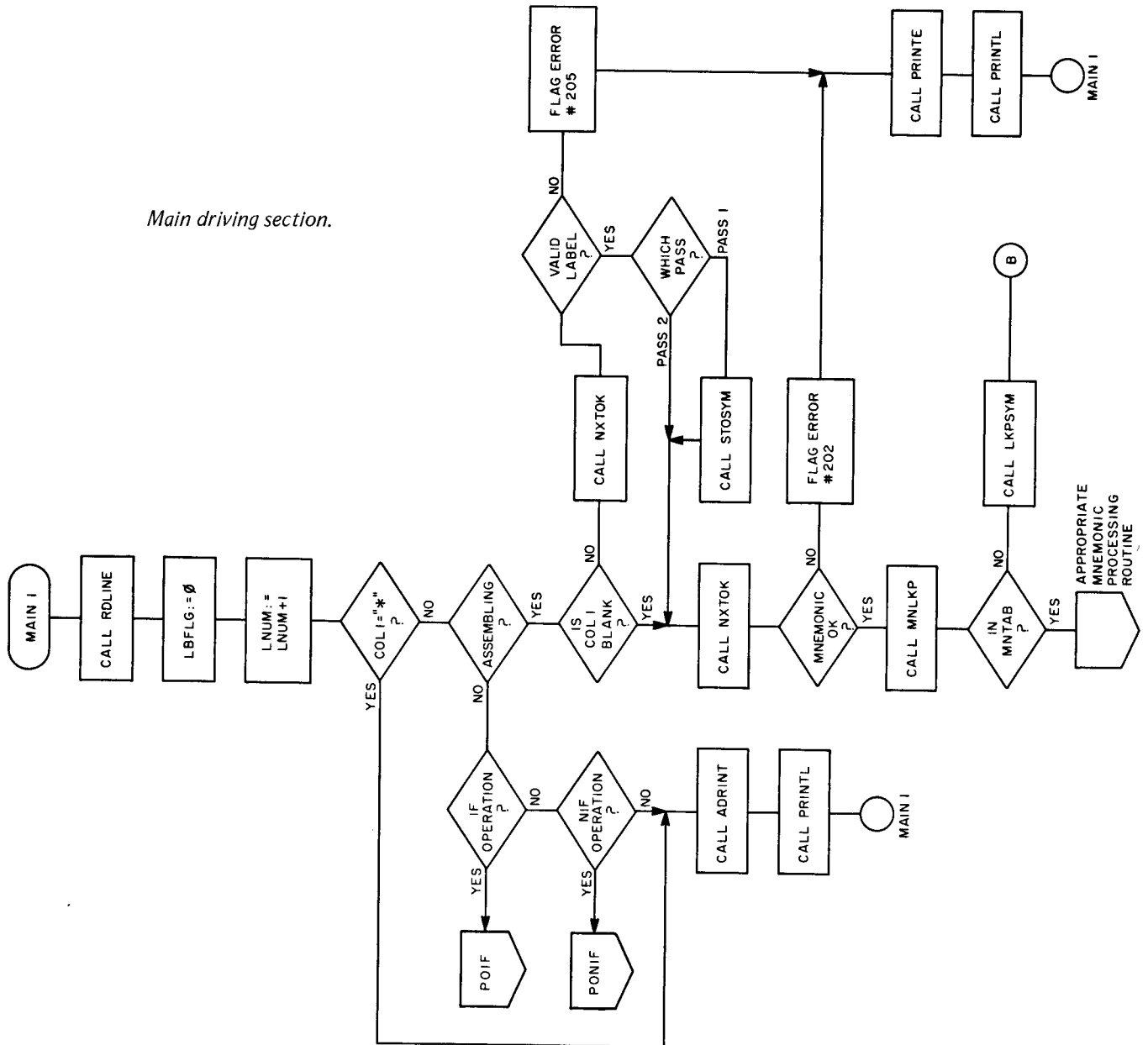
The memory layout of this common block shows the different symbols applied to corresponding bytes in PROG1 and PROG2:

COMMON BLOCK

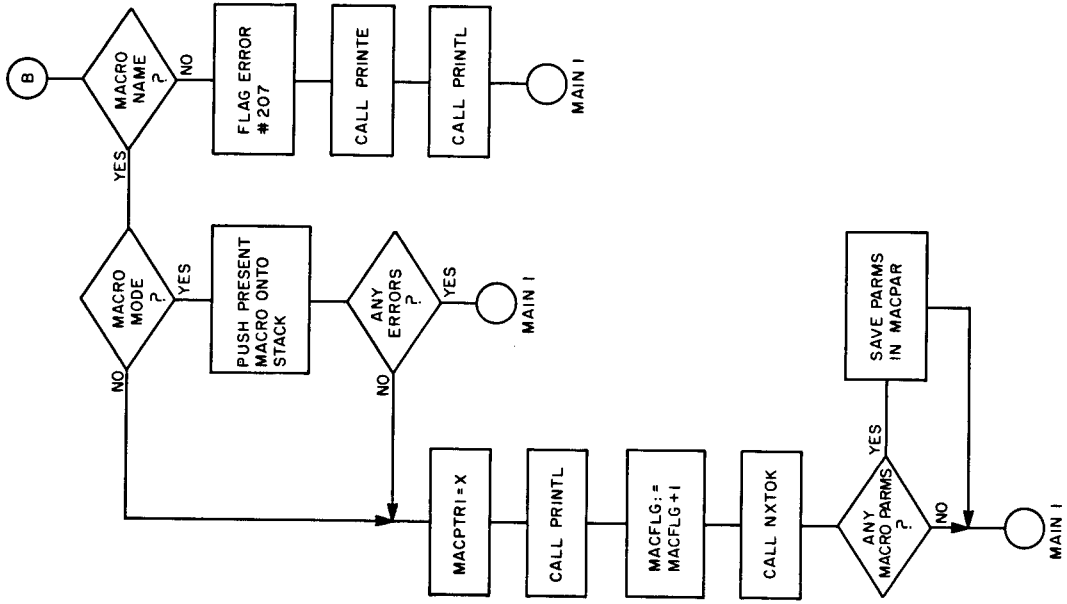
	PROG1	PROG2
0	AAA	DDD
1	AAA + 1	DDD + 1
2	AAA + 2	EEE
3	AAA + 3	EEE + 1
4	AAA + 4	FFF
5	BBB	GGG
6	BBB + 1	GGG + 1
7	BBB + 2	GGG + 2
8	BBB + 3	GGG + 3
9	BBB + 4	GGG + 4
10	BBB + 5	GGG + 5
11	BBB + 6	GGG + 6
12	BBB + 7	GGG + 7
13	BBB + 8	GGG + 8
14	BBB + 9	GGG + 9
15	CCC	GGG + 10
16	CCC + 1	GGG + 11
17	CCC + 2	GGG + 12
18	CCC + 3	GGG + 13
19	CCC + 4	GGG + 14
20	CCC + 5	GGG + 15
21	CCC + 6	GGG + 16
22	CCC + 7	GGG + 17
23	CCC + 8	GGG + 18
24	CCC + 9	GGG + 19

Example 3: Entry Points and External References				
Here we show two programs which refer to symbols in each other using ENT and EXT pseudo operations to establish linkages.				
* NAM PROG1				
	ENT SUB1	DEFINE SUB1 AS AN ENTRY POINT		
	ENT SUB2	DEFINE SUB2 AS AN ENTRY POINT		
* SUB1 LDX #\$0000				
	.			
	.			
	RTS			
* SUB2 LDA B# 'C				
	.			
	.			
	RTS			
	END			
NAM PROG2				
	EXT SUB1	DEFINE SUB1 AS EXTERNAL		
	EXT SUB2	DEFINE SUB2 AS EXTERNAL		
* JSR SUB1 CALL SUB1 IN PROG1				
	JSR SUB2	CALL SUB2 IN PROG2		
	.			
	.			
	END			

Example 3: This example shows some of the linkage possible between external routines using the ENT and EXT pseudo instructions. The routine PROG1 is shown here with two entry points, SUB1 and SUB2. PROG2 defines these labels as EXTERNAL and references them with a jump instruction.■



Main driving section.



The Assembler

Assembler Modules Overview

With this section a detailed description of the inner workings of the Assembler begins. As stated previously, this is a two pass assembler: pass 1 is used to determine values and resolve all references (labels, externals, etc.); pass 2 generates and outputs relocatable machine code, prints listings and messages. This is all controlled from the main program module MAIN1.

Main

MAIN1 is the driving section or top level of the Assembler. It is in one of two logical states, Pass 1 or Pass 2. The state is reflected in the value of system variable PASS. If PASS has the value hexadecimal 00 then the Assembler is executing Pass 1. If PASS has the value hexadecimal FF then the Assembler is executing Pass 2.

Calls: ADRINT, LKPSYM, MACPSH, MNLKP, NXTOK, PRINTE, PRINTL, RDLIN, STOSYM
Flags: IFFLG, LBFLG, MACFLG, PASS
Pointers: CUCHAR, CULINE, DESCRA, DESCRC, MACPTR
Temporaries: MACSAV
Buffers: MACPAR

Pass 1 of Main

The purpose of this pass is to assign a location to each data defining pseudo operation and to each instruction and thus, to assign a value to labels (symbols) appearing in the *label* fields of the input source program. To facilitate this a Location Counter (LC) is kept. This counter contains the address of the first byte of the line currently being processed. The Location Counter is initialized to hexadecimal 0000 at the beginning of Pass 1 and is incremented at the end of the processing of an instruction. The value of the increment is equal to the number of bytes the instruction just processed requires.

Pass 1 proceeds by reading a line of source code from the input file. The *label* field is then scanned to see if there

is a label present. If there is, the label (symbol) is stored in the Symbol Table (SYMTAB) along with the value of the Location Counter.

The next field scanned is the *opcode* field. A search of the Mnemonic Table (MNTAB) is done to find the address of the processing routine that handles the mnemonic opcode found. MNTAB contains this address along with part of the machine code for processing the opcode. The rest of the machine code is calculated by the processing routine. For the pseudo operations the machine code part of the entry in MNTAB is ignored.

When a mnemonic opcode is found in MNTAB, the address of the processing routine is extracted, the partial machine code is loaded into a register, and control is passed to the processing routine for the mnemonic found.

If the search of MNTAB was unsuccessful in locating the particular opcode, a search of the Symbol Table (SYMTAB) is made to see if the mnemonic is the name of a Macro.

If the mnemonic is a Macro call then the value of the symbol found in the Symbol Table is the address of the Macro definition in the Macro Table (MACTBL). A flag (MACFLG) is set and control is passed back to the main loop. This switches the pointers so that lines of source code now come from the Macro Table rather than from the input file. When the end of the Macro is identified, MACFLG is cleared and the lines of source code come once again from the input file.

Because the number of bytes that an operation or pseudo instruction requires is dependent on the *operand* field, PASS1 must evaluate the *operand* field to determine the increment that is to be added to the Location Counter.

When processing is complete for that opcode, control is passed back to the main program loop and another line of source code either from the input file or Macro Table is read and processed. When the pseudo instruction END is encountered, PASS1 finishes up by requesting that the input source file be rewound and jumping to PASS2.

Pass 2 of Main

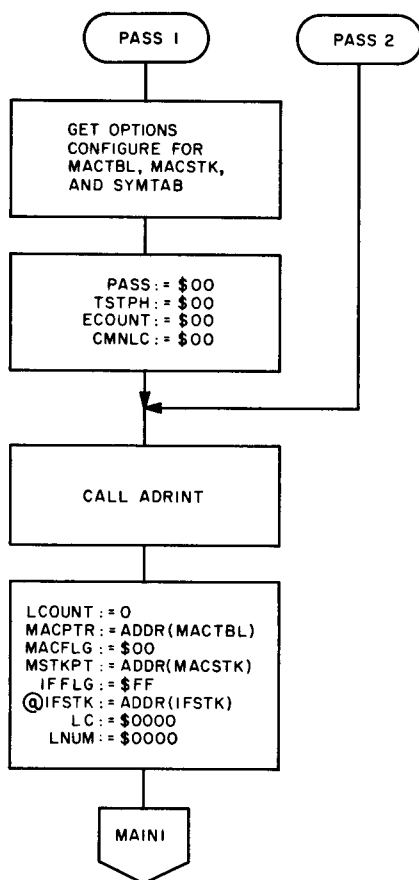
The purpose of PASS2 is to generate and output the machine code, print listings (if selected by the user) and print

any error messages found in the input source program. (PASS1 also prints some error messages.)

PASS2 proceeds through the same main loop as PASS1; however, when control is passed to the processing routines, they calculate the machine code and output it, whereas PASS1 did not. As noted earlier the Assembler can tell what pass it is executing by testing the one byte flag called PASS.

The output listing is unbuffered and is printed line by line after a statement is processed.

When the END pseudo instruction is encountered in the source code the Assembler writes out any machine code that is in the output buffer, prints the Symbol Table (if that option was selected by the user), and terminates by passing control back to the system monitor.



PASS1 and PASS2 initialization. ADDR(Y) indicates the address of item Y.

Tables

There are four tables used by the Assembler: MNTAB, SYMTAB, CHRTAB, and MACTBL.

MNTAB AND CHRTAB are permanent tables and SYMTAB AND MACTBL are constructed by the Assembler.

Mnemonic Table (MNTAB)

MNTAB is the table that contains the valid machine mnemonics and pseudo instructions recognized by the Assembler. Each entry in the table is six bytes long. The format is:

CCCXY

where:

CCC = 3 byte mnemonic;
XX = 2 byte address of the processing routine for this instruction;
Y = 1 byte part of the machine code for this instruction. The other part of the machine code is calculated by the processing routine in PASS2. This field is ignored by the pseudo operation processing routines.

Symbol Table (SYMTAB)

SYMTAB is the symbol table and is maintained with access by means of a hash code. Each entry is 9 bytes long. The format is:

CCCCCXXF

where:

CCCCC = 6 byte symbol. If a symbol is less than 6 bytes long, blanks are inserted on the right.
XX = 2 byte address or value of the symbol.
F = 1 byte of flags.

bit 7 Redefined flag
bit 6 Relocation flag
bit 5 Macro flag
bit 4 Common flag
bit 3 External flag
bit 2 Entry flag
bit 1 Reserved for future use
bit 0 Reserved for future use

If a bit is set (1) it means that the associated symbol has that bit position's attribute. If the symbol is a Macro, the XX above is the address location of the Macro definition in the Macro Table (MACTBL). The length of SYMTAB is calculated based on the length of the Assembler and the Macro Table. It is approximately 4 K bytes long, enough for over 300 symbols. This length can be changed by modifying the Symbol Table initialization routine in the main program of the Assembler.

Character Table (CHRTAB)

CHRTAB is used by the lexical analysis routines to facilitate the classification of characters. Each recognizable ASCII character value hexadecimal 20 to 5F is in the table.

The table is indexed by using the value of the ASCII character plus the base address of the table.

The definition of the individual bits in the single byte entry are:

bit 7 Alpha character
bit 6 Numeric character
bit 5 Arithmetic operator

bit 4	Location separator
bit 3	Mnemonic separator
bit 2	Operand separator
bit 1	Hexadecimal character
bit 0	A, B, or X register character

Macro Table (MACTBL)

The Macro Table (MACTBL) is the location where the actual Macro definition code is stored. The size of MACTBL is approximately 2 K bytes. Its organization is free form; ie: one Macro could be anywhere from one instruction to 2 K bytes long. A pointer in the Symbol Table keeps track of where the Macro definition begins, and another MAC pseudo instruction signals the end of the previous Macro definition. The length of the MACTBL is a function of the length of the Assembler, the length of the Symbol Table, and the overall length assumed required for execution of the Assembler (16 K). This length can be modified by changing the table initialization routine in the main program of the Assembler.

Stacks

If Stack (IFSTK)

This is a stack used by the IF and NIF pseudo instruction processing routines to allow the nesting of the IF and NIF pseudo instructions. Eight levels of nesting are allowed.

Macro Stack (MACSTK)

This stack is used to allow the nesting of Macro calls. The number of allowed calls varies depending upon the number of parameters on each Macro. A maximum of 35 levels is possible if no parameters are on the nested calls. A minimum of about four levels is the limit if the maximum number of parameters is used on each nested Macro call. Its actual length is 100 bytes. Note that while Macro *definitions* cannot be nested at all, *expansions* can be nested within these limits when one Macro references another Macro within its definition.

Utility Routines

These utility routines perform comparisons, additions, conversions, etc., as needed by other routines. They operate on numeric or alphabetic data depending on their specific function.

COMPAR

This routine is used to compare variable length character strings or variable length byte strings. The string lengths can be up to 255 bytes. When COMPAR is called the Index register X points to a parameter list of 5 bytes:

Bytes 1, 2..... Address of first string
 Bytes 3, 4..... Address of second string
 Byte 5..... Number of bytes to be compared.

On returning from COMPAR the result of the comparison is reflected in the 6800's condition codes register. For example, a typical call might look like the following:

LDX --- (pointer to parameter list) ---
 JSR COMPAR
 BNE NOMATCH taken if string 1 is not equal to string 2
 BEQ MATCH taken if string 1 equals string 2

Calls: none
 Called By: MNLKP, POEND, POMAC, SYMCMP
 Flags: none
 Pointers: Index Register
 Temporaries: XSAV

CVHB

This routine converts hexadecimal character strings into a sixteen bit binary value. When CVHB is called location DESCRA contains the address of the hexadecimal string, and location DESCRC contains the string length. The length cannot exceed four. The converted value is returned in the Index register.

Calls: CVHBS
 Called By: NSEVL
 Flags: none
 Pointers: DESCRA, DESCRC
 Temporaries: HVAL

CVHBS

This routine is used by the CVHB routine to convert an ASCII hexadecimal character to binary. On entry the Index register points to the character and on return the A register contains the binary value.

Calls: none
 Called By: CVHB

CVDB

This routine converts decimal character strings into a sixteen bit binary value. When CVDB is called location DESCRA contains the address of the string and location DESCRC contains the length. The length cannot exceed five. The converted value is returned in the Index register.

Calls: MPY16
 Called By: NSEVL
 Pointers: DESCRA, DESCRC
 Temporaries: DCOUNT, DVAL, DXSAV, TENVL

CVBTD

This routine converts a sixteen bit binary value into a five character decimal string. On entry, registers A and B contain the sixteen bit binary value to be converted and the Index register points to an area of storage where the converted string is to be stored.

Calls: none
 Called By: POEND, PRINTE
 Pointers: Index register
 Temporaries: SAVEA, SAVEX, SAVEX1

ADD16

This routine adds together two unsigned sixteen bit values. On entry, the Index register points to a four byte area of storage that contains the values to be added together. Bytes 1 and 2 are added to bytes 3 and 4 and the result is stored in bytes 1 and 2.

Calls: none
Called By: GCHRTB, HASH, MNLKP, NSEVL

SUB16

This routine subtracts two unsigned sixteen bit values. On entry, the Index register points to a four byte area of storage that contains the values to be subtracted. Bytes 3 and 4 are subtracted from bytes 1 and 2 and the result is stored in bytes 1 and 2.

Calls: none
Called By: NSEVL

MPY16

This routine multiplies two unsigned sixteen bit values. On entry, the first value is in registers A and B and the second value is in the two bytes pointed to by the Index register.

The result is truncated to sixteen bits and returned to registers A and B.

Calls: none
Called By: CVDB, HASH, MNLKP, NSEVL

DIV16

This routine divides two unsigned sixteen bit values. On entry, the dividend is in registers A and B and the divisor is in the two bytes pointed at by the Index register. The result is placed into registers A and B, and the remainder is returned in the Index register.

Calls: none
Called By: HASH, NSEVL

Listing Routines

The following section includes routines used to output print lines for listings and messages in their proper formats (see listings 1 and 2). These routines utilize the input and output routines outlined in the section **Input and Output Routines** to do the actual detail IO functions.

PRINTL

This routine checks the options byte during Pass 2 to see if the L option and the M option have been selected. PRINTL calls routine OUTL to print a line of listing if the L option has been selected. PRINTL also checks to see if the Assembler is in the Macro mode; if it is, it checks the M option to see if expansion lines from macros are to be listed.

Calls: LINCK, OUTL, SPACER
Called By: ADDR9, LCNAB1, LCN2, LCN3, MAIN,

POCMN, POEND, POENT, POEQU, POEXT,
POFCB, POFCC, POFDB, POIF, POMAC,
PONIF, PORMB
MACFLG, OPTNS, PASS

Flags:

OUTL

This routine does the actual printing of the listing. On entry, the following system global values are used to format the listing:

MCOUNT	Number of bytes of machine code (0, 1, 2, or 3)
POP	Pseudo instructions to be printed 0,1, or 2 bytes.
OPCD	Opcode in hexadecimal to be printed
ADR1,ADR2	Second and third bytes of machine code
LINEN	Line number
MACFLG	Macro mode flag
CMNFLG	Common flag
RELFLG	Relocatable flag
ENTFLG	Entry flag
EXTFLG	External flag

For all flags, hexadecimal 00=no, and FF=yes

Calls: OUT2HS, OUT4HS, OUTCHR, PDATA1, PRINTL
Called By: PRINTL
Entries: OUTL7A (from PRINTE)

PRINTE

This routine prints error messages on the system console. Error messages are always printed as the errors occur during both PASS1 and PASS2. On entry, the Index register contains the error number in a binary coded decimal format. The routine prints the error number and the source line that caused the error, then increments the error count in ECOUNT. This count is printed at the very end of the assembly.

Calls: CVBTD, OUTL7A, PDATA1
Called By: ADDR1-5,7,8, INXCK, LBLCK, MACMOV, MAIN, POCMN, POEND, POENT, POEQU, POEXT, POFDB, POFCC, POFDB, POMAC, PONAM, PORMB, P2ERR, RDMAC, STOSYM
Pointers: ECOUNT
Temporaries: ERNUM

LINCK

This routine is called to make sure that the output listing is formatted into pages of 60 lines each. If the line count (LCOUNT) is equal to zero, the system console is spaced to the top of the next page.

Calls: SPACER
Called By: POEND, PRINTL
Pointers: LCOUNT

SPACER

This routine performs the above spacing and also prints a

Listing 1: Output listing format. A sample of the Assembler listing option showing the format of a program listing.

- ① Columns 1 to 4; line number generated by the Assembler.
- ② Column 5; plus sign (+) if this line is a Macro expansion, blank otherwise. Column 6; blank.
- ③ Columns 7 thru 10; the hexadecimal memory location. Column 11; blank.
- ④ Columns 12 and 13; the hexadecimal operation code. Column 14; blank.
- ⑤ Columns 15 thru 18; the operand, either 2 or 4 hexadecimal characters. Column 19; blank.
- ⑥ Column 20; type indicator: Relocatable (R), Macro (M), Entry (E), External (X), or Common (C). Column 21; blank.
- ⑦ Columns 22 thru 72; the first 51 characters of the source statement. Note that the source statement is not reformatted.
- ⑧ This line of periods acts as a logical page separator. It is repeated every 66 lines, preceded by 3 blank lines and followed by 2 blank lines. Thus there are 60 lines of code possible per page. This page separator is provided for those whose printers use roll paper, and will result in 11 inch pages if line spacing is 66 lines per inch.
- ⑨ If the Statement is a comment (designated in the source by an asterisk (*) in the first column of the source), then columns 1 thru 5 (① and ② above) are the same as above, columns 6 thru 21 are blank, and columns 22 thru 72 are again the first 51 characters of the source statement (in this case, the comment).

```

① 140V 0966 CE 093E R      LDX  #ENSIZ  GET ENTRY LENGTH
    1410 0969 5A          DEC B      B:=IP-1
    1411 096A BD 0B7U R      JSR  MPY1A  GET ((IP-1)*6
    1412 096D B7 07E3 R      STA A  PSING1  SAVE
    1413 0970 F7 07E4 R      STA B  PSING1+1
    1414 0973 CE 0006 R      LDX  #MNTAB
    1415 0976 FF 07E5 R      STX  PSING2  PSING2:=BASE OF MNTAB
    1416 0979 CE 07E3 R      LDX  #PSING1  POINT TO PARMS
    1417 097C BD 0BEC R      JSR  ADD16  PSING1:=(IP-1)*6+MNTAB
    1418 097F FE 07E3 R      LDX  PSING1
    1419 0982 FF 07EB R      STX  TBADD  SAVE
    1420
    1421 * COMPARE MNEMONIC WITH ENTRY IN MNTAB
    1422
⑧ .....
    ②
    ⑥
    1423 0985 FE 027B R      LUX  DESCRA  GET MNEMONIC ADDRESS
    1424 0988 FF 07E5 R      STX  PSING2  INIT PARM FOR COMPARE
    1425 098B CE 07E3 R      LDX  #PSING1  POINT TO PARMS
    1426 098E B3 06C5 R      JSR  COMPAR  COMPARE
    1427 0991 25 0B          BCS  MNLI  ENTRY<MNEMONIC
    1428 0993 26 11          BNE  MNMI  ENTRY>MNEMONIC
    1429
    1430 0995 4F          CLR A      ENTRY FOUND
    1431 0996 FE 07E8 R      LDX  TBADD  POINT TO ENTRY
    1432 0999 E6 05          LDA B 5,X  GET MC
    1433 099B EE 03          LDX  3,X  GET BRANCH ADDRESS
    1434 099D 39          RTS
    1435
    1436 * ENTRY<MNEMONIC LP:=IP
    1437 *
    1438 099E B6 093D R      MNLI  LDA A IP
    1439 09A1 B7 093B R      STA A LP
    1440 09A4 20 A9          BRA  MNLKPA  TRY AGAIN
    1441
    1442 * ENTRY>MNEMONIC MP:=IP
    1443 *
    1444 09A6 B6 093D R      MNMI  LDA A IP
    1445 09A9 B7 093C R      STA A MP
    1446 09AC 20 A1          BRA  MNLKPA  TRY AGAIN
    ⑦

```

Listing 2: Symbol table listing format. A sample of the Assembler listing option showing the format of a symbol table listing.

- ① Columns 1 thru 6; the symbol name. Column 7; blank.
- ② Columns 8 thru 11; the hexadecimal address at which the symbol is defined. Column 12; blank.
- ③ Column 13; the symbol type: Relocatable (R), Macro (M), Entry (E), External (X), or Common (C).
- ④ This line of periods acts as a logical page separator. It is repeated every 66 lines, preceded by 3 blank lines and followed by 2 blank lines. Thus there are 60 lines of table entries possible per page. This page separator is provided for those whose printers use roll paper, and will result in 11 inch pages if line spacing is 66 lines per inch.

```

OUTS  1872 R
P2ERR 1007 R
P2ERRA 10E3 R
P2ERRB 10E9 R
PAGEA 17B2 R
PAGEU 17BB R
PASS  0275 R
PASS1 038E R
PASS2 0467 R
PBLK2 143D R
PBLOCK 143B R
PBXS  144F R
PCOUNT 07E7 R
PJATA1 185E RN
PJATA2 185A R
PLEND 0C29 R
POCMN  1104 R
POCMN0 11E1 R
POCMN1 11E4 R
POCMN2 11E9 R
POCMN3 122F R

```

```

① .....
    ②
    ④
    ③
POCMN4 1246 R
POEND  124E R
POEND0 1254 R
POEND2 1268 R
POENT  13D8 R
POENT1 13ED R
POENT2 1401 R
POENT3 1426 R
POENT4 1435 R
POEQU  1451 R
POEXT  14AC R
POEXT1 14C4 R
POEXT2 14E0 R
POEXT3 1508 R
POEXT4 150B R
POFCB  150E R
POFCC  1543 R
POFDB  159A R
POIF  15EE R
POIFA 15FB R
POIFB 1603 R
POIFC 1621 R
POIFE 1626 R
POMAC 162E R
POMAC1 164F R
POMAC2 166F R
POMAC5 168F R
POMAC6 16A1 R
POMAC7 16D0 R
POMAC8 16CA R
POMACA 16E0 R
PONAM  1723 R
PONAM1 1739 R
PONAM2 1744 R
PONIF  1758 R
POP  0C66 R
POPAG 179D R
PORMB 17BE R

```

series of periods that are at convenient 11 inch intervals (assuming 6 lines per inch) to allow the listing to be torn off and put into a page size notebook.

Calls: PDATA1
Called By: LINCK, PRINTL

Mnemonic and Symbol Table Routines

The following routines provide the table look-up, comparison, and insertion functions necessary for maintenance of the Mnemonic and Symbol Tables.

MNLKP

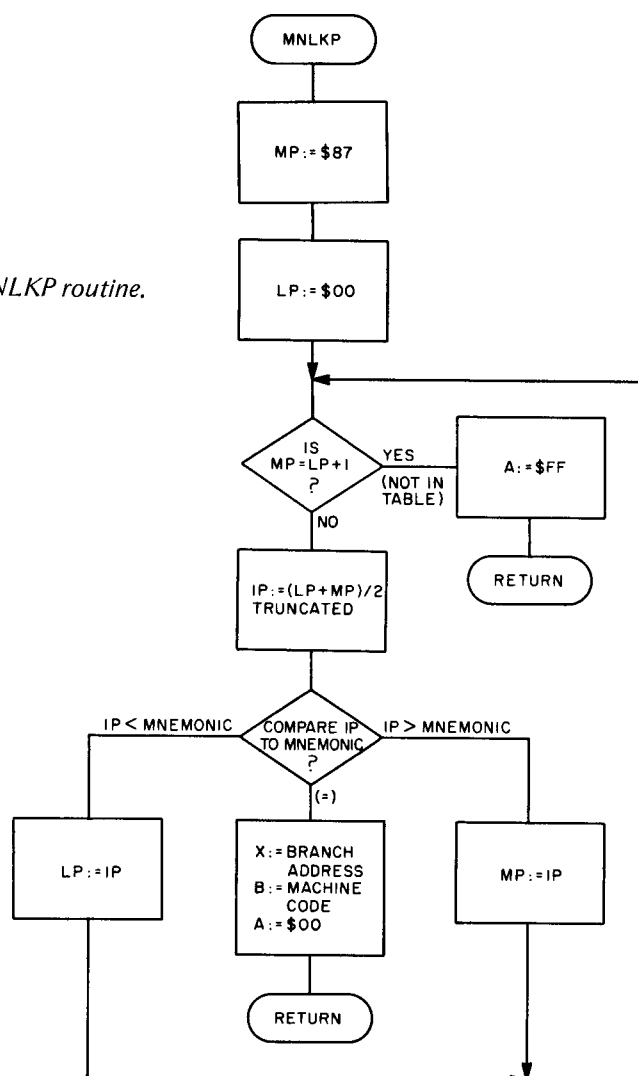
This routine is used to search the Mnemonic Table (MNTAB). On entry, DESCRA points to the mnemonic opcode to be searched for, and DESCRC contains the

length of the mnemonic opcode (3). On return register A contains a return code. Hexadecimal FF is the return code if the mnemonic is not in MNTAB. Hexadecimal 00 is the return code if the mnemonic is in the table, and on return register X contains the processing routine's address and register B contains the partial opcode.

The algorithm used is a binary search in which the interval to be searched is divided into two nearly equal parts, the part which does not contain the searched for item is discarded, and the part which contains the sought item is similarly processed until the wanted item is located. The binary search is used because it is significantly faster than a linear search.

Calls: ADD16, COMPAR, MPY16
Called By: MAIN
Pointers: DESCRA, DESCRC, PCOUNT, PSTNG1, PSTNG2, TBADD
Temporaries: ENSIZ, IP, LP, MP

Flowchart of MNLKP routine.



STOSYM

This routine is used to store a symbol and its value into the hash coded Symbol Table (SYMTAB). Hash coding is the method by which the actual symbol that is to be stored in the table is used to find the address of the Symbol Table entry. Hashing means simply to hash or to jumble up the bits of the ASCII characters in a symbol in such a way that a fairly unique number is generated. This number, after further manipulation, becomes the actual address of the location in the Symbol Table where the symbol is to be stored (located).

On entry, DESCRA contains the address of the symbol (from the *label* field) to be stored and DESCRC contains the length. Routine HASH is called to create a hashed code to access the table. If the entry at this address, called the probe address, is empty then the symbol and its value is

stored there and the routine returns. If the entry at this probe address is not empty then a new probe must be calculated. This is done by looking at the next sequential address after the first probe to see if it is empty. If it is the symbol and its value are stored at this new location. If this new entry is also occupied then the next sequential address after this new probe is checked, etc. This manner of rehashing is called the Linear Rehash method.

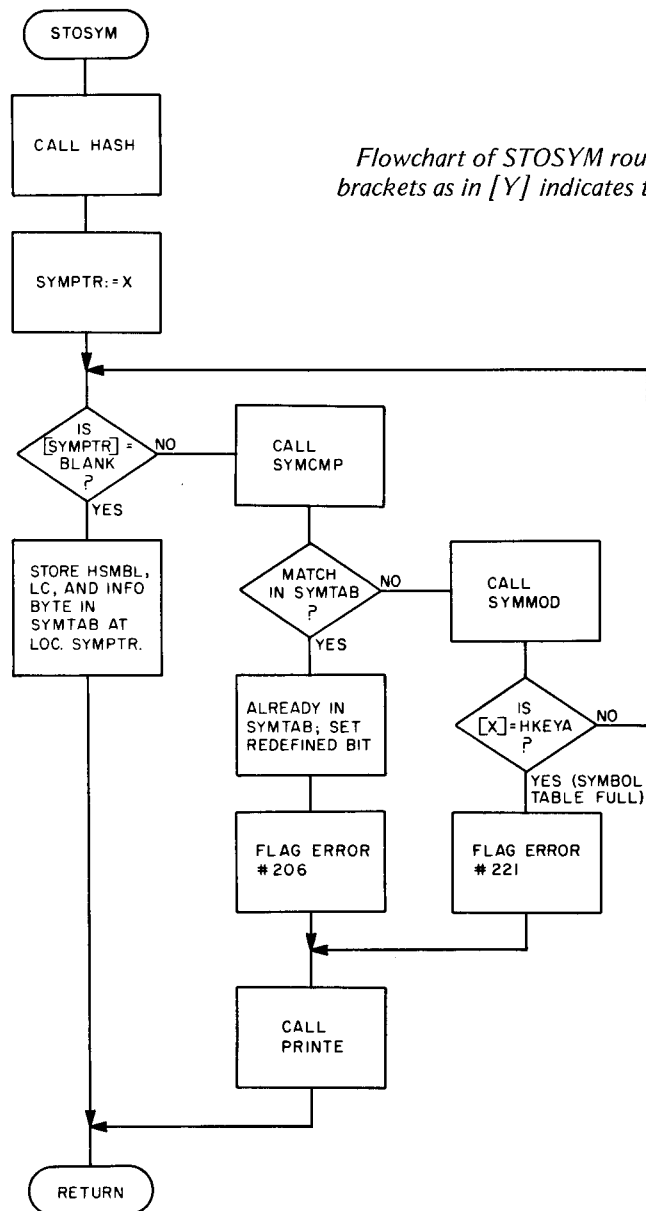
If the symbol is already in the Symbol Table then an error message is printed and the routine returns. If the entire table is found to be full then an error message is printed and the routine returns.

Calls: HASH, PRINTE, SYMCMP, SYMMOD

Called By: MAIN, POCMN, POEXT, PONAM

Pointers: SYMPTR

Temporaries: HSAV1, HSAV2, HSMBL



LKPSYM

This routine is used to look up a symbol in the Symbol Table. On entry, DESCRA contains the address of the symbol to be looked up and DESCRC contains the length. This routine proceeds much as the STOSYM routine except that if the symbol is found the value of the flag byte is returned in register B.

If the symbol is not found in the Symbol Table then a return code of hexadecimal FF is returned in register B.

Calls: HASH, SYMCMP, SYMMOD

Called By: MAIN, NSEVL, POCMN, POENT, POEXT

Pointers: HKEYA, SYMPTR

HASH

This routine is used to create a hashed code for accessing the Symbol Table (SYMTAB). On entry, DESCRA contains

the address of the symbol to be hashed and DESCRC contains the length.

The hashed value is calculated by first folding over the six bytes of the symbol (spaces are added to the right of symbols less than six bytes long) into two bytes by adding the six bytes together in groups of two bytes. This value is divided by the maximum number of symbols that the Symbol Table can hold (NSYM). The remainder from this division is then multiplied by nine (the entry length) and the base address of the Symbol Table is added to produce a pseudo random address. This value is returned in the Index register.

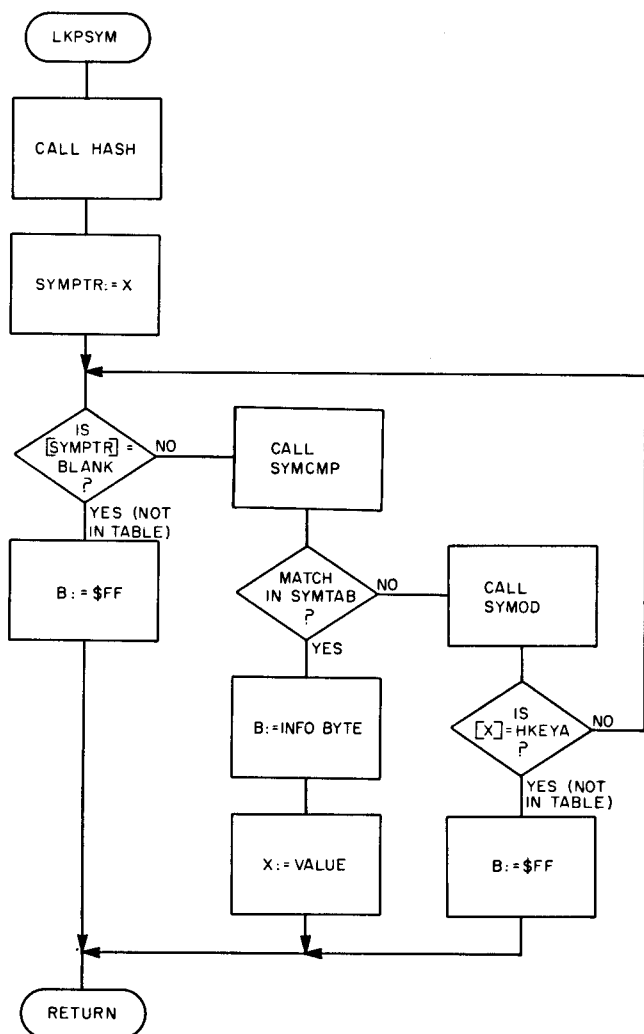
Calls: ADD16, DIV16, MPY16

Called By: LKPSYM, STOSYM

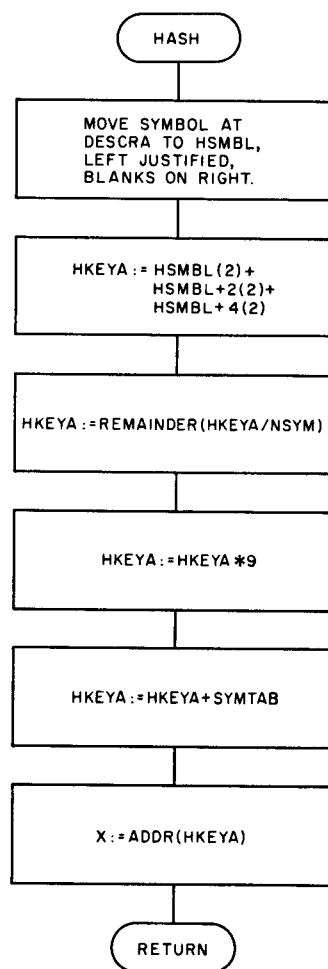
Pointers: DESCRA, DESCRC, NSYM

Temporaries: HKEYA, HKEYB, HSAV1, HSAV2, HSMBL

Flowchart of LKPSYM routine. The use of square brackets as in [Y] indicates the contents at address Y.



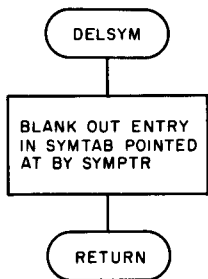
Flowchart of HASH routine. ADDR(Y) indicates the address of item Y.



DELSYM

Sometimes it is necessary to delete an entry from the Symbol Table. This routine does the deletion but it can only delete the last entry that has been added to the Symbol Table. On entry, SYMPTR contains the location of the last symbol stored and this is the entry that is deleted.

Calls: none
Called By: LBLCK
Pointers: SYMPTR



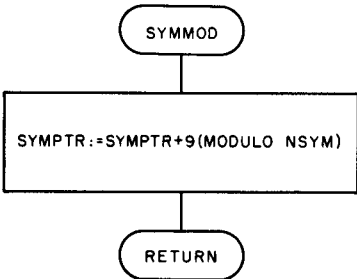
Flowchart of DELSYM routine.

SYMCMP

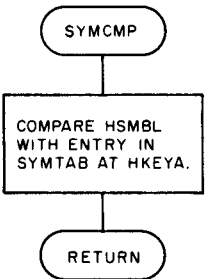
This routine is used to compare a symbol with an entry in the Symbol Table. On entry, the Index register points to the entry in the Symbol Table and HSMBL contains the symbol.

On return, the results of the comparison are reflected in the condition codes (see COMPAR).

Calls: COMPAR
Called By: LKPSYM, STOSYM
Pointers: HSMBL, PCOUNT, PSTNG1, PSTNG2



Flowchart of SYMMOD routine.



Flowchart of SYMCMP routine.

Input and Output Routines

These IO routines perform the details of formatting information from and to the particular medium used for the source and object code. These routines are independent of the particular serial medium used to store the code. The routines which are directly dependent on the type of medium are described in the section *Interfacing and Using the Assembler*.

OUTBNR

This routine stores the single ASCII character in register B into the output file.

The character may be an:

- “R” — Relocatable
- “N” — Entry
- “X” — External
- “M” — Common

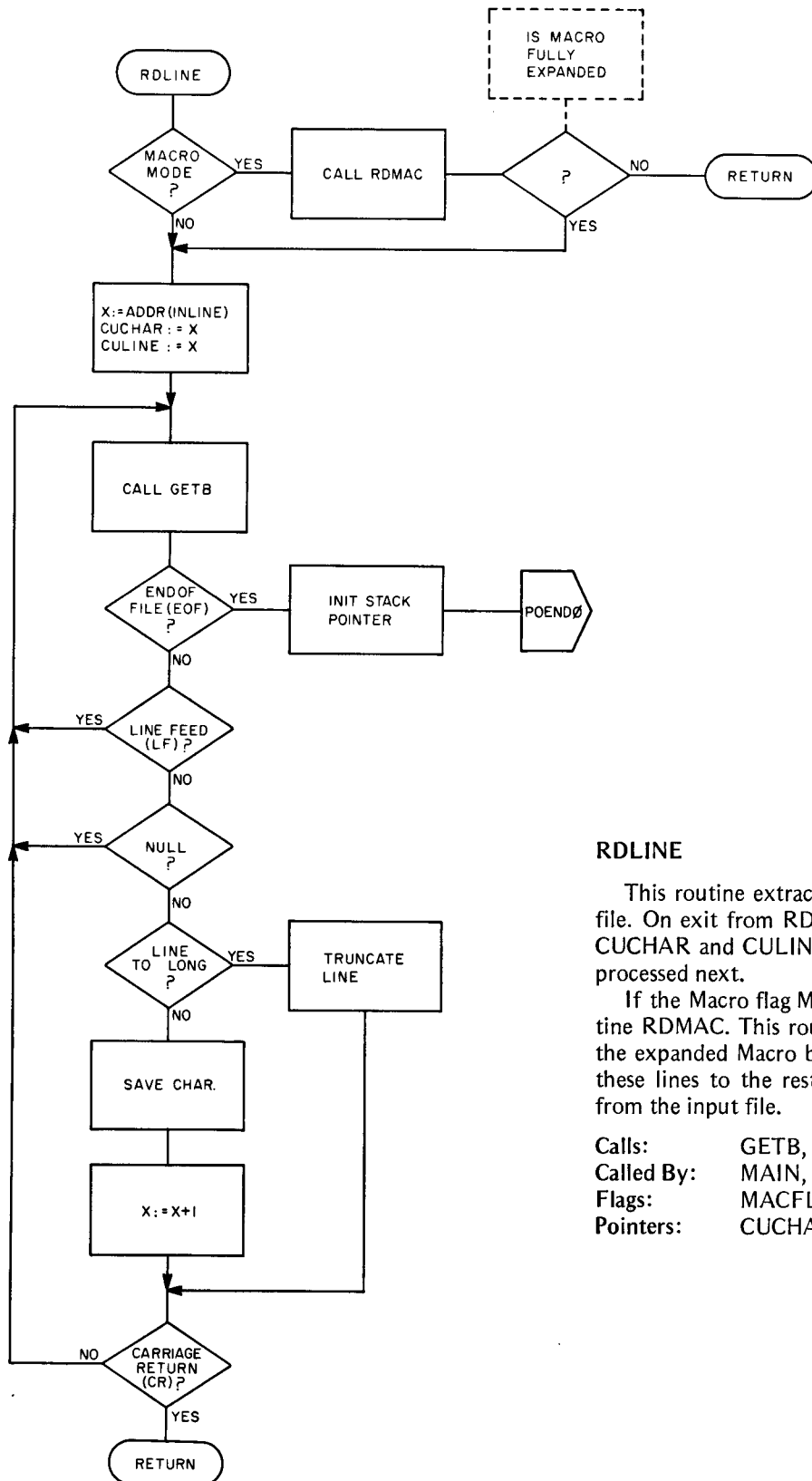
Calls: OUTB
Called By: POENT, POEXT, POFDB, PONAM
Flags: OPTNS

OUTBIN

This routine puts machine code into the output file. On entry, register B contains one byte of machine code. OUTBIN translates this byte into two bytes of ASCII hexadecimal characters and then calls OUTB to output the two bytes to the object file.

Calls: OUTB, OUTHL, OUTHR
Called By: ADDR9, LCNAB1, LCN2, LCN3, PBLOCK, POENT, POEXT, POFDB, POFCC, POFDB, PONAM, RMBOUT
Flags: OPTNS

Flowchart of RDLINE routine. ADDR(Y) indicates the address of item Y.



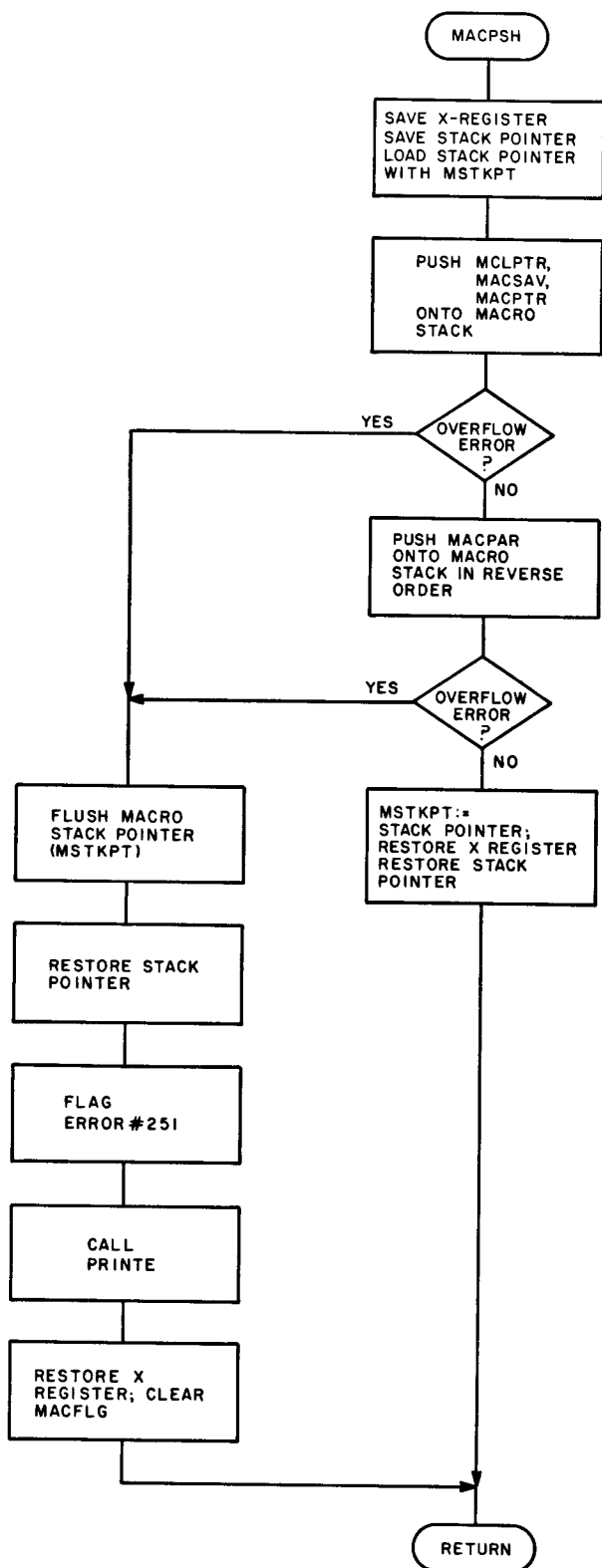
RDLINE

This routine extracts lines of source code from the input file. On exit from RDLINE, the Assembler global pointers CUCHAR and CULINE point to the input line that is to be processed next.

If the Macro flag MACFLG is set, RDLINE calls the routine RDMAC. This routine passes lines of source code from the expanded Macro back to RDLINE and RDLINE passes these lines to the rest of the Assembler, rather than lines from the input file.

Calls: GETB, RDMAC
 Called By: MAIN, POMAC
 Flags: MACFLG
 Pointers: CUCHAR, CULINE, INBUF

Flowchart of MACPSH routine.



MACPSH

This routine is used when a nested Macro is called. The state of the present or outer Macro being expanded is pushed onto the Macro Stack (MACSTK).

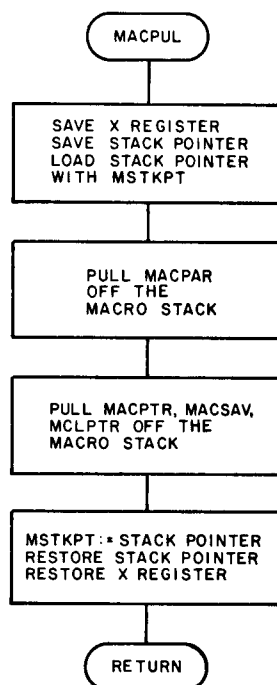
Calls: PRINTE
Called By: MAIN
Flags: MACFLG
Pointers: MACEND, MACPTR, MACSAV, MACSTK, MCLPTR, MSTKPT
Temporaries: STKSAV, MXSAV1, MXSAV2
Buffers: MACPAR

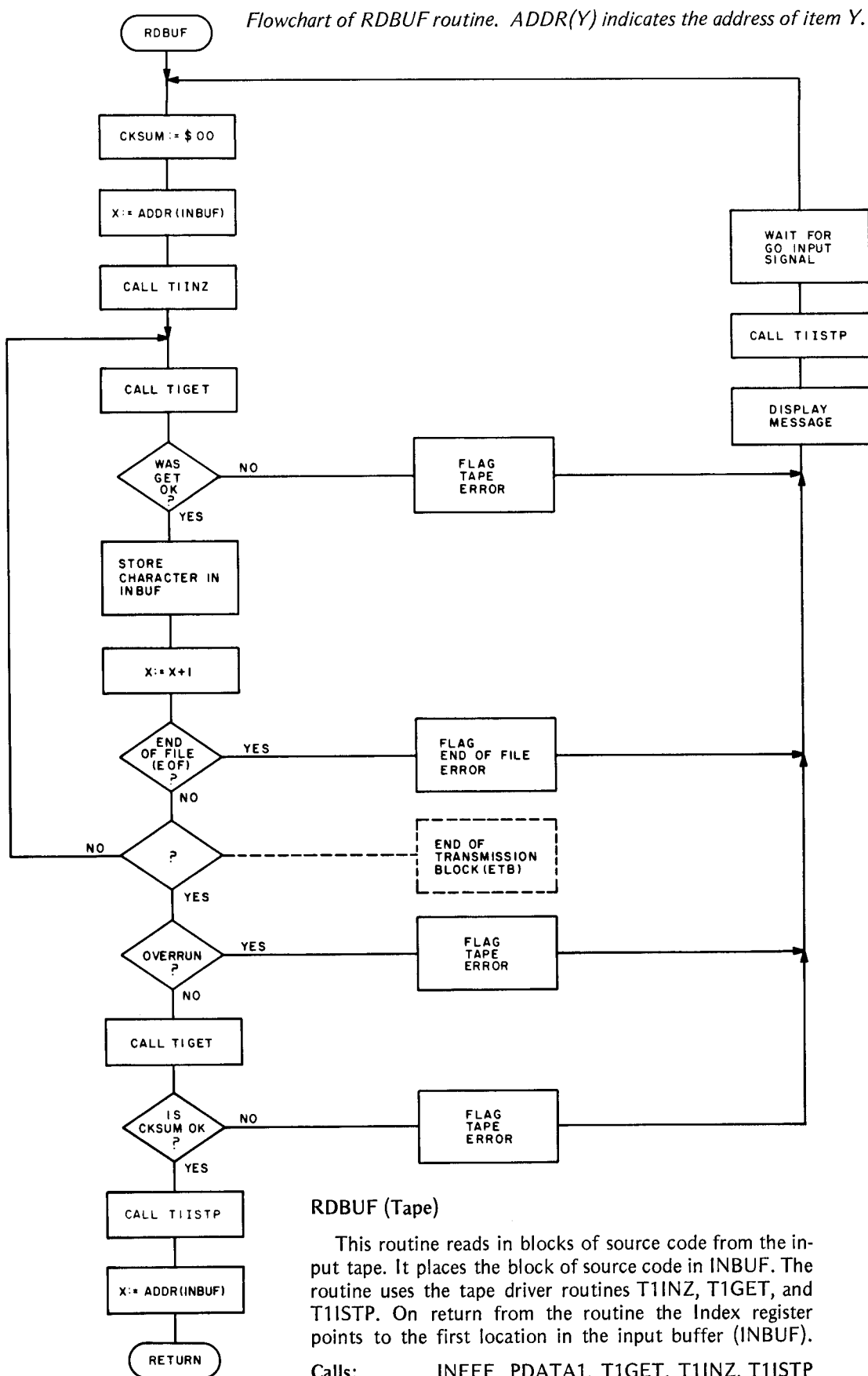
MACPUL

This routine is used to pull the state of a Macro previously pushed onto the Macro Stack. This is done when the inner Macro has been fully expanded.

Calls: none
Called By: RDMAC
Pointers: MACPTR, MACSAV, MCLPTR, MSTKPT
Temporaries: MXSAV1, STKSAV
Buffers: MACPAR

Flowchart of MACPUL routine.



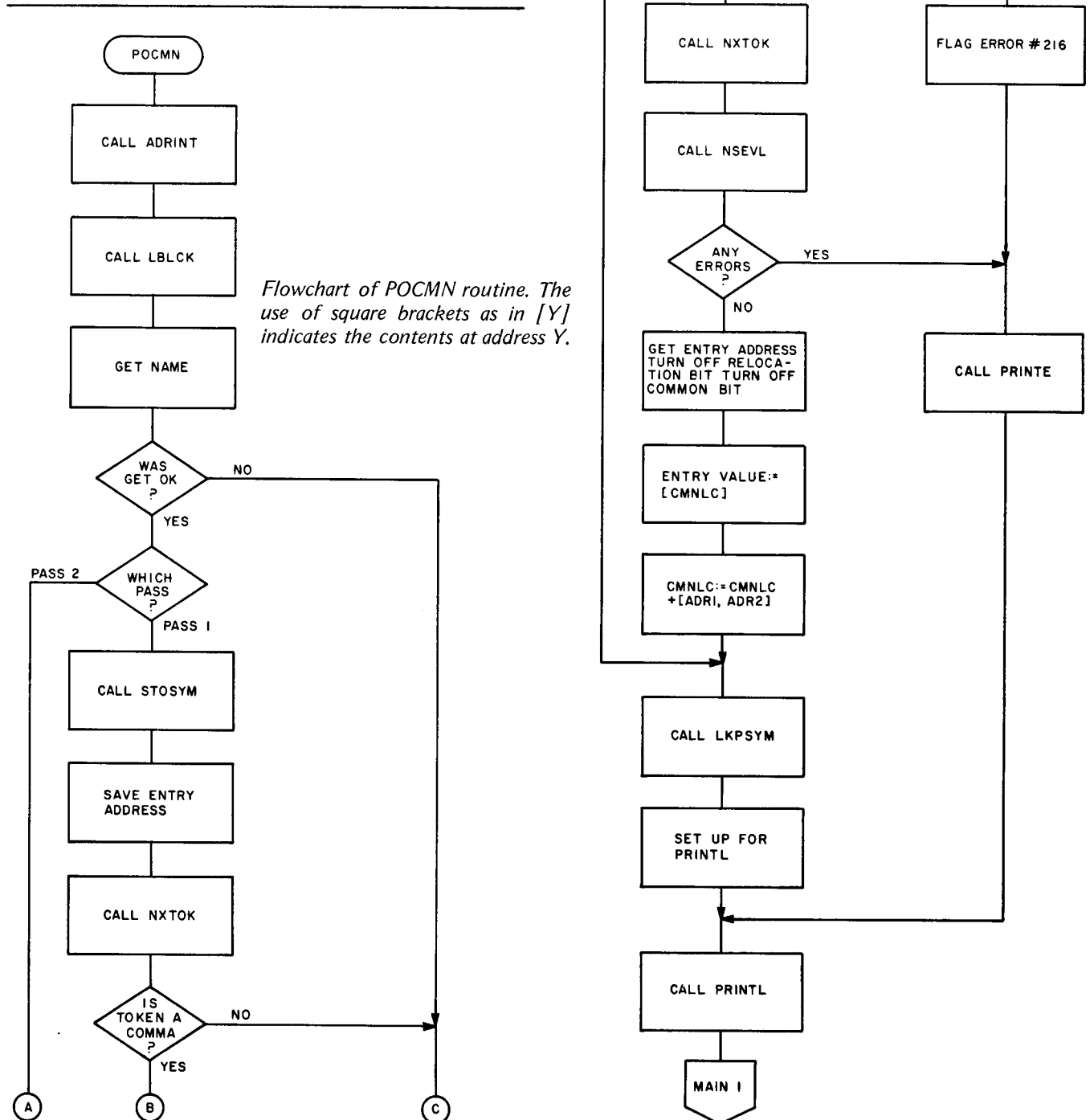


Pseudo Instruction Processing

Following are the descriptions of the routines which process the set of pseudo instructions defined for this Assembler (see "Pseudo Instructions" in the section **The Source Language**).

POCMN

This routine processes the CMN pseudo operation. In Pass 1, the name in the *operand* field is stored in the Symbol Table with: REL bit set equal to off (0), Common bit set equal to on (1), Value set equal to value of the Common Location Counter (CMNLC).



CMNLC is then incremented by the number of bytes allocated by the second operand.

In Pass 2 there is very little processing done by POCMN other than setting up flags for the listing line.

Calls: ADRINT, LBLCK, LKPSYM, NSEVL, NXTOK, PRINTE, PRINTL, STOSYM

Called By: MAIN

Flags: CMNFLG, MCOUNT, PASS, POP

Pointers: ADR1, ADR2, CMNLC, SYMPTR

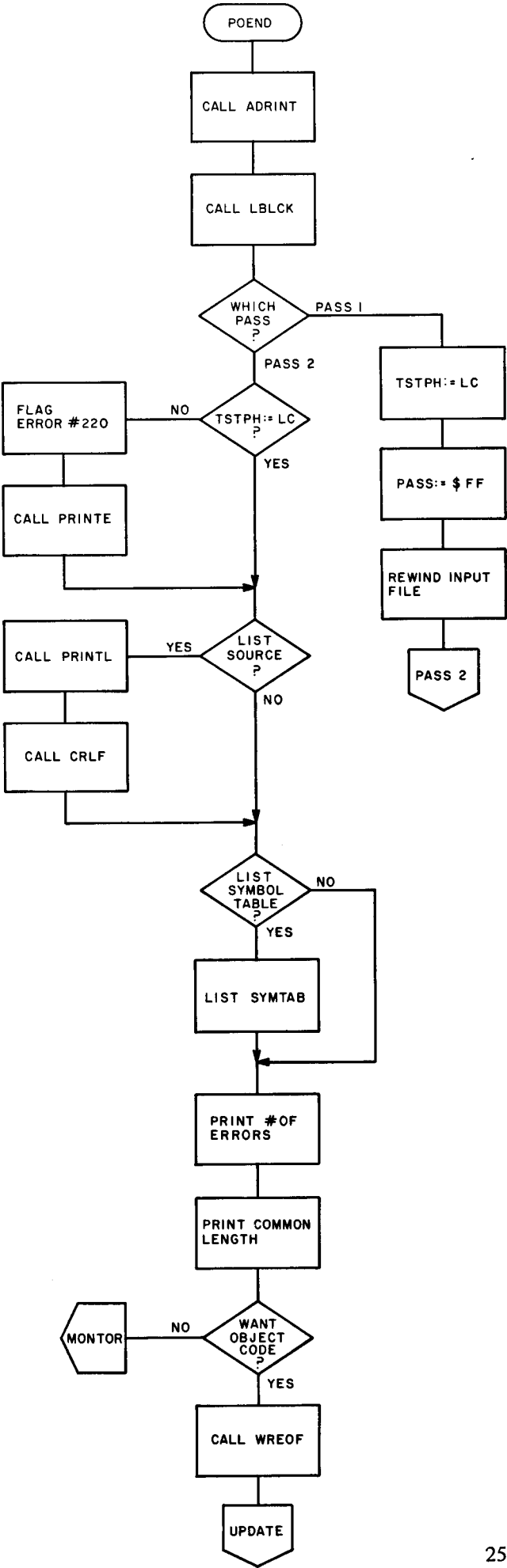
Temporaries: CMNXS

POEND

This routine processes the END pseudo operation. In Pass 1, POEND initializes the system global value TSTPH. TSTPH is used by the Assembler to check for phasing errors. A phasing error is one in which an instruction is at different locations during Pass 1 and Pass 2. This can occur, for example, if a Macro definition follows the call to that Macro.

During Pass 1 POEND also sets PASS to Pass 2, rewinds the input file, and then transfers control to PASS2. In Pass 2, POEND finishes up the assembly by printing the Symbol Table, if selected, printing the error count, closing the output file, and transferring control to the system monitor.

- Calls: ADRINT, COMPAR, CRLF, CVBTD,
 LBLCK, LINCK, OUTCHR, PDATA1,
 PRINTE, PRINTL, RESTR, WREOF
- Called By: MAIN
- Flags: OPTNS, PASS, SORTF
- Pointers: CBLOCK, CMNLC, CXS2, LC, PCOUNT,
 PSTING1, PSTING2, TSTPH, ZZZ
- Temporaries: ENDXS



Flowchart of POEND routine.

POENT

This routine processes the ENT pseudo operation. During Pass 1 the statement is checked for syntax. In Pass 2 the symbol in the *operand* field is looked up in the Symbol Table. The ENT bit in the entry in the Symbol Table is then set. The symbol, the entry address, "R", and "N" are output to the output file, providing linking information to the Linking Loader.

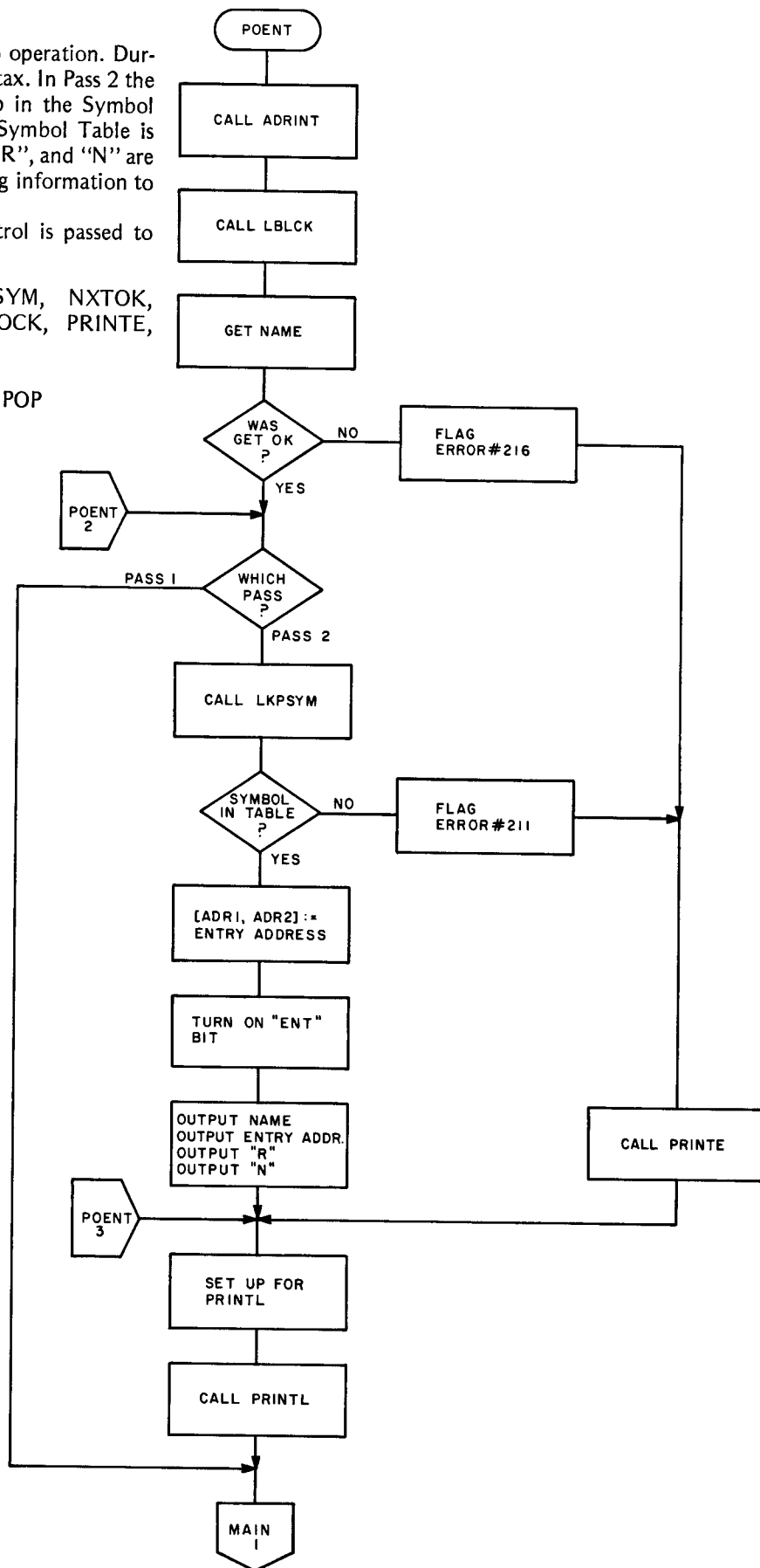
Following Pass 1 or 2 processing control is passed to entry point MAIN1.

Calls: ADRINT, LBLCK, LKPSYM, NXTOK, OUTBIN, OUTBNR, PBLOCK, PRINTE, PRINTL

Called By: MAIN

Flags: ENTFLG, MCOUNT, PASS, POP

Pointers: ADR1, ADR2, SYMPTR



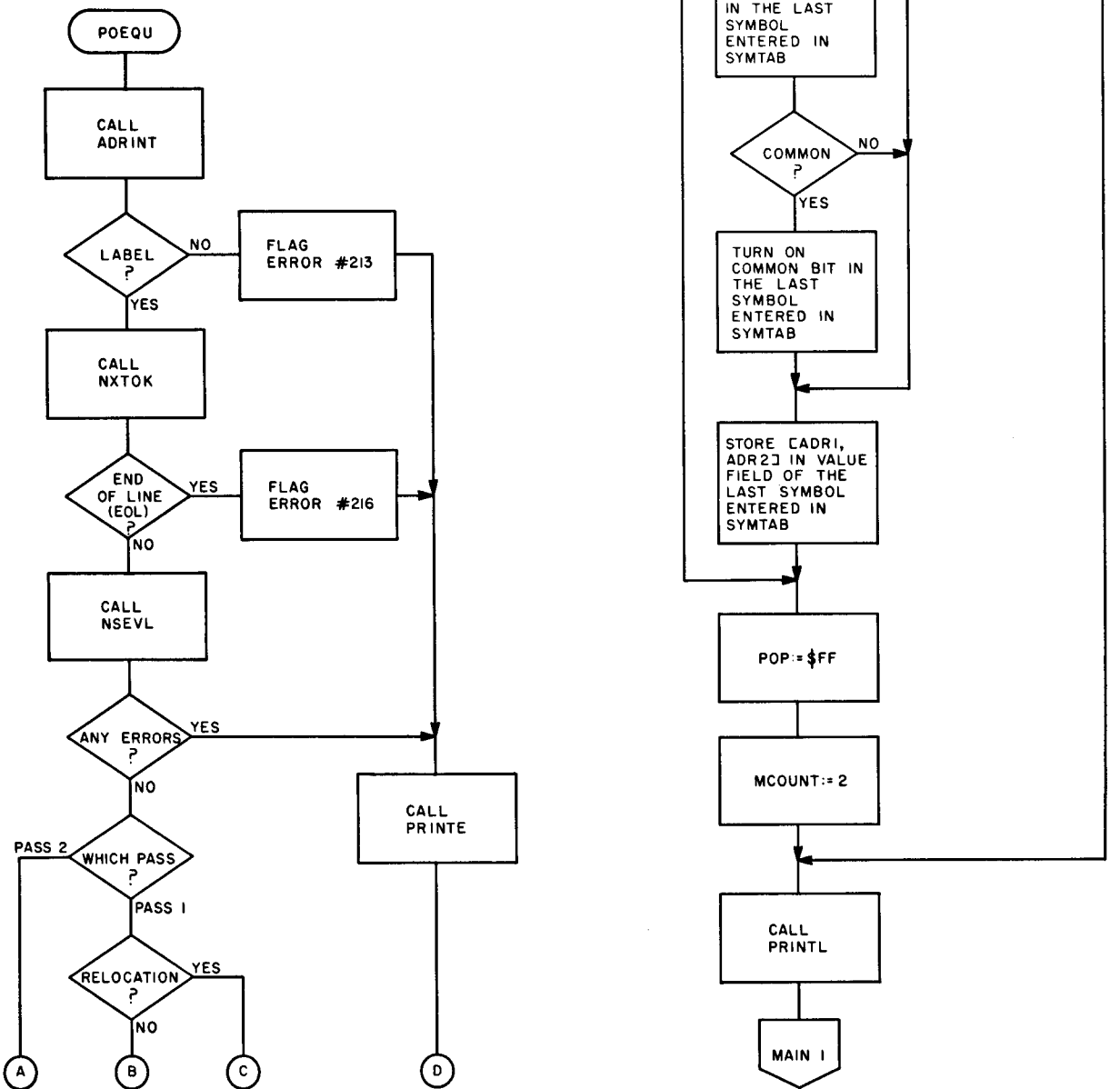
Flowchart of POENT routine. The use of square brackets as in [Y] indicates the contents at address Y.

POEQU

This routine processes the EQU pseudo instruction. In Pass 1 the operand is evaluated and stored in the Symbol Table entry associated with the label on the EQU statement. During Pass 2 there is no processing done other than printing the line of listing.

Following Pass 1 or 2 processing control is passed to entry point MAIN1 in the main loop.

Calls: ADRINT, NSEVL, NXTOK, PRINTE, PRINTL
Called By: MAIN
Flags: LBFLG, MCOUNT, PASS, POP, RELFLG
Pointers: SYMPTR
Temporaries: EQUXS



Flowchart of POEQU routine. The use of square brackets as in [Y] indicates the contents at address Y.

PBLOCK

This routine is used by the POENT and POEXT routines to output the Entry/External symbol to the output file.

Calls: OUTBIN
Called By: POENT, POEXT
Pointers: SYMPTR
Temporaries: PBXS

POEXT

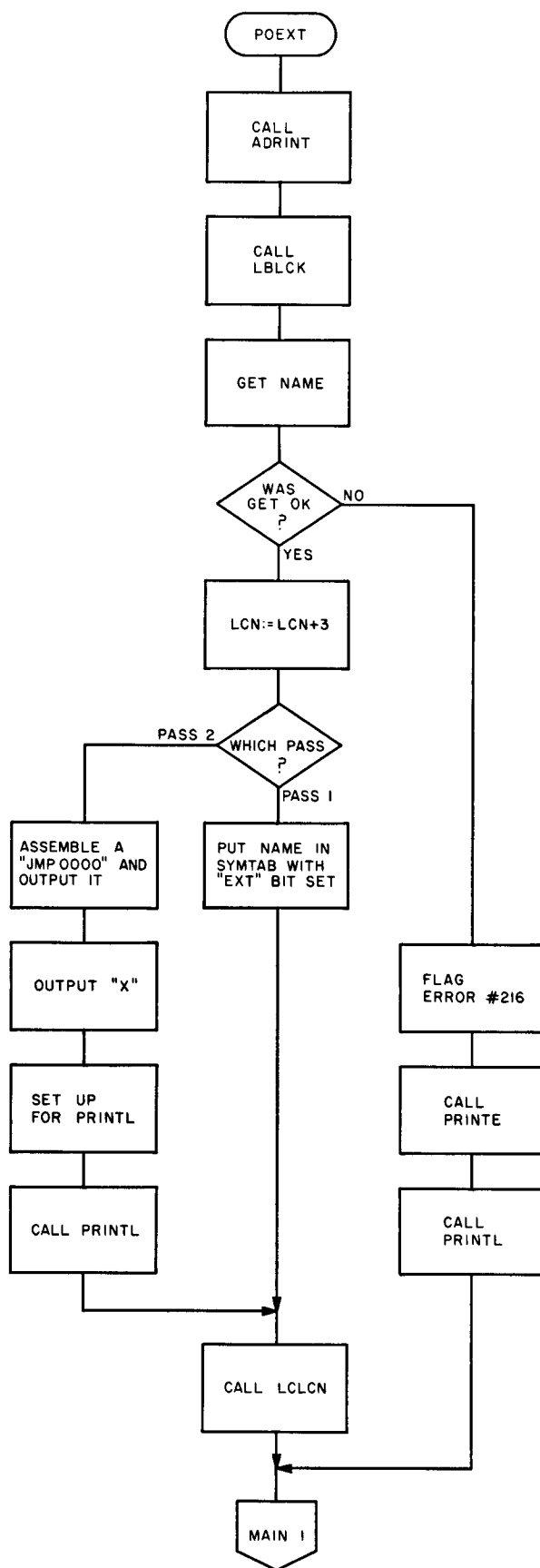
This routine processes the EXT pseudo operation. In Pass 1 the symbol in the *operand* field of the statement is stored in the Symbol Table with the EXT bit set and a value equal to the current value of the location counter.

In Pass 2, a JMP symbol (3 bytes) is assembled and output to the output file, along with the EXT indicator "X". This provides linking information to the Linking Loader.

Following Pass 1 or 2 processing control is passed to entry point MAIN1.

Calls: ADRINT, LBLCK, LCLCN, LKPSYM, NXTOK, OUTBIN, OUTBNR, PBLOCK, PRINTE, PRINTL, STOSYM
Called By: MAIN
Flags: EXTFLG, MCOUNT, PASS
Pointers: ADR1, ADR2, LCN, OPCODE, SYMPTR

Flowchart of POEXT routine.



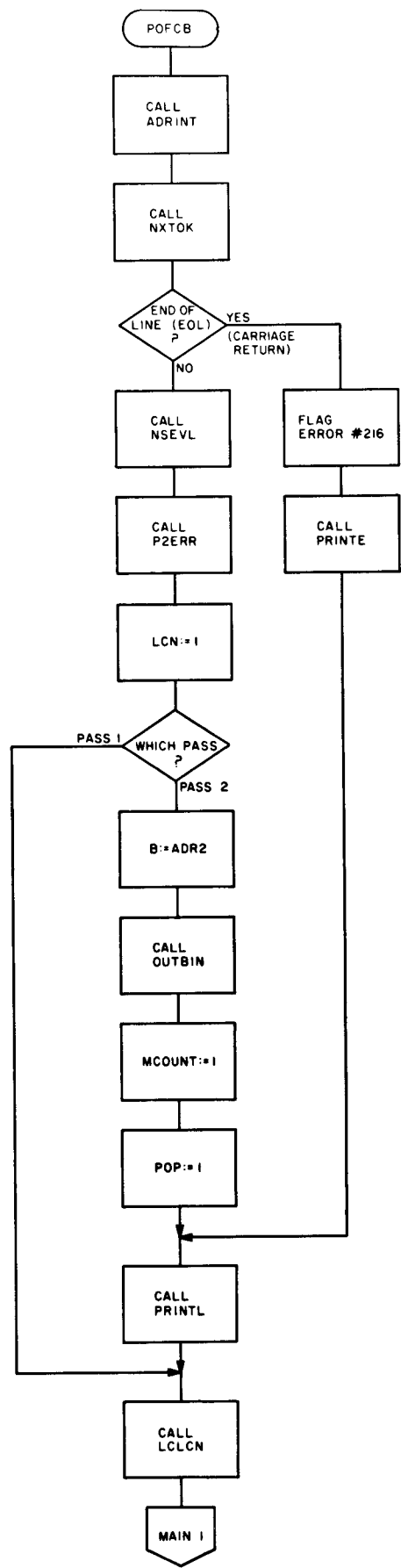
POFCB

This routine processes the FCB pseudo instruction. During Pass 1, POFCB simply increments the Location Counter (LC).

In Pass 2 the LC is incremented as in Pass 1, but the operand is evaluated and stored as a one byte value in the output buffer.

Following Pass 1 or 2 processing control is passed to entry point MAIN1 in the main loop.

Calls: ADRINT, LCLCN, NSEVL, NXTOK,
 OUTBIN, PRINTE, PRINTL
Called By: MAIN
Flags: MCOUNT, PASS, POP
Pointers: LCN



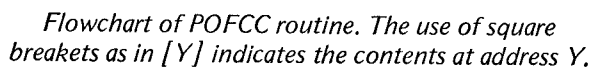
Flowchart of POFCB routine.

This routine processes the FCC pseudo instruction. In Pass 1 the Location Counter is incremented by the number of characters in the operand of the statement.

Following Pass 1 or 2 processing control is passed to entry point MAIN1 in the main loop.

Called By: MAIN

Pointers: ADR1, ADR2, CUCHAR, LCN



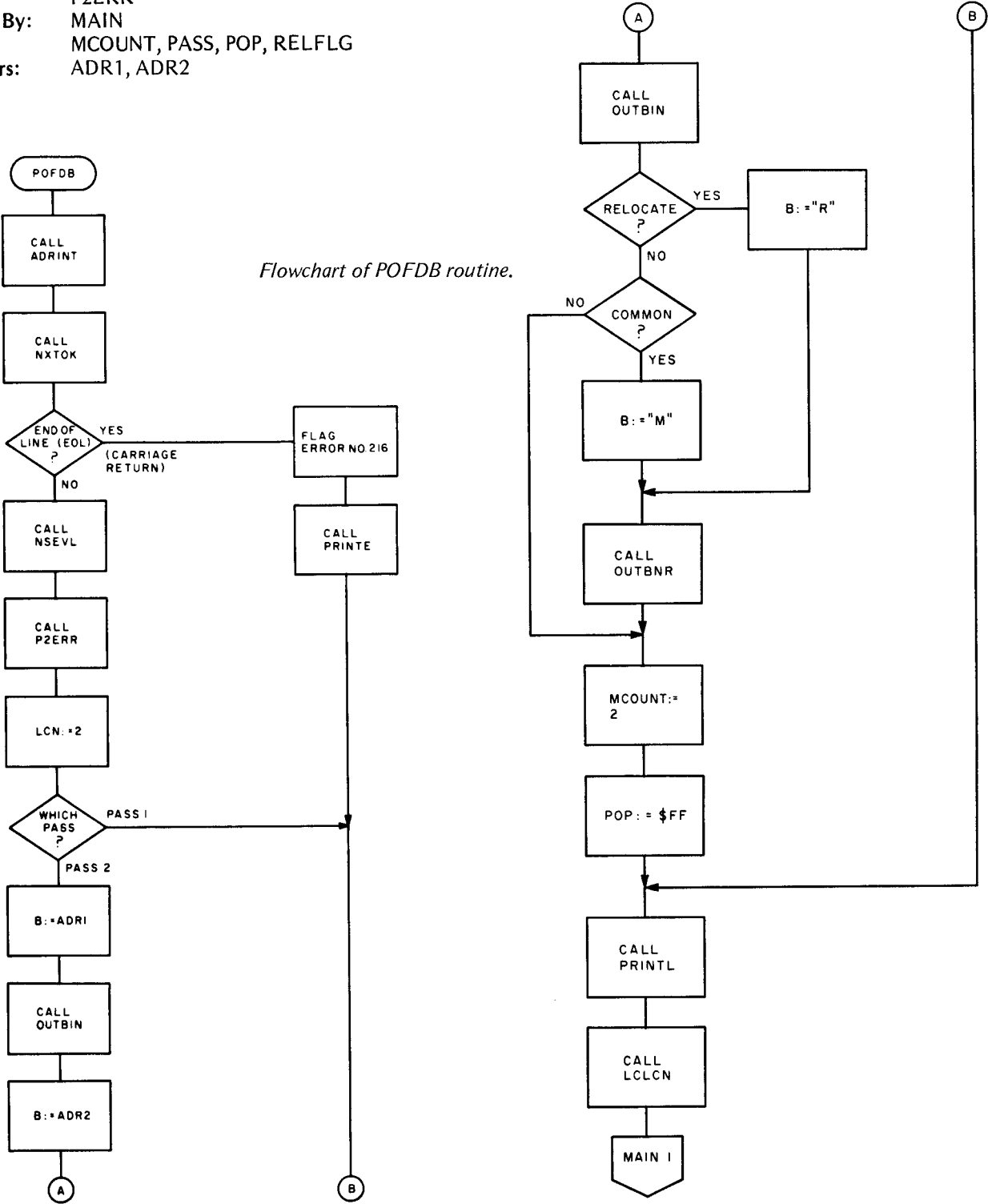
POFDB

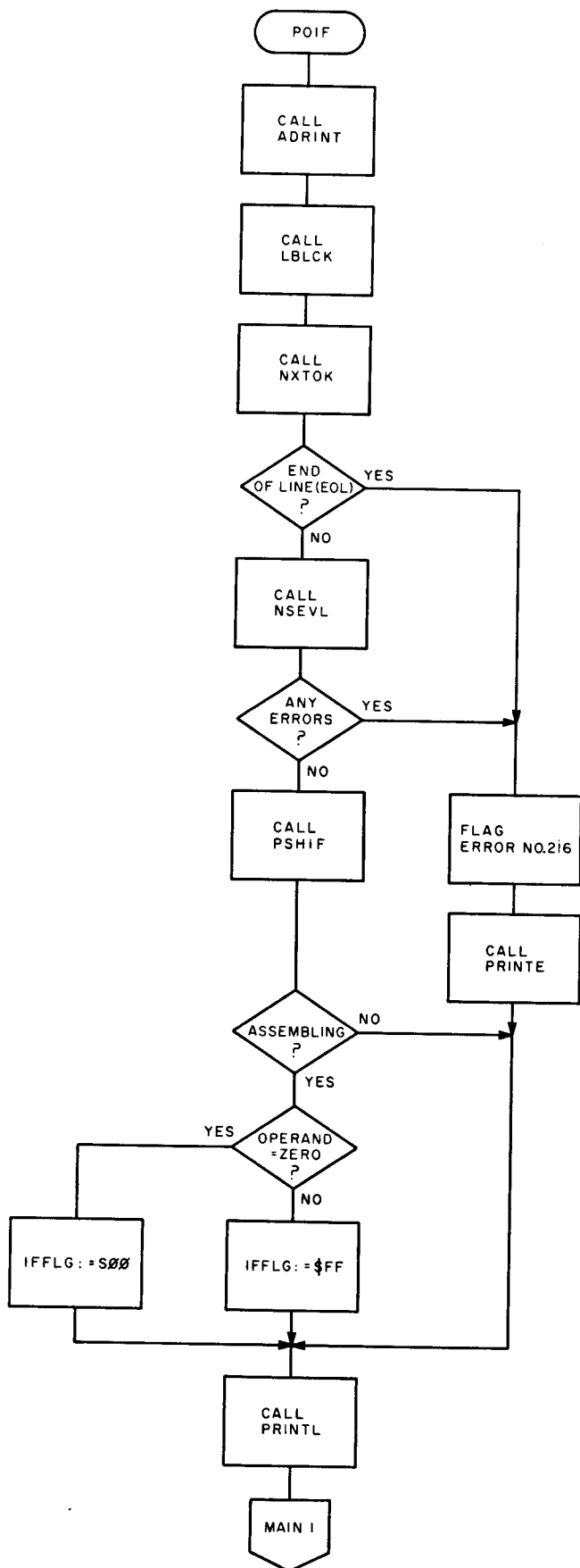
This routine processes the FDB pseudo operation. It is almost the same as routine POFDB, except that two byte values are used instead of one byte values.

Following Pass 1 or 2 processing control is passed to entry point MAIN1 in the main loop.

Calls: ADRINT, LCLCN, NSEVL, NXTOK,
 OUTBIN, OUTBNR, PRINTE, PRINTL,
 P2ERR
Called By: MAIN
Flags: MCOUNT, PASS, POP, RELFLG
Pointers: ADR1, ADR2

Flowchart of POFDB routine.





POIF

This routine processes the IF pseudo instruction. There is no distinction made between Pass 1 and 2.

The operand is evaluated and the present value of the flag IFFLG is stacked in IFSTK. Then depending on whether the operand value is 0 or not, IFFLG is set or cleared.

Cleared: Not assembling

Set: Assembling

Calls: ADRINT, LBLCK, NSEVL, NXTOK, PRINTE, PRINTL, PSHIF

Jumps: MAIN1

Called By: MAIN

Flags: IFFLG

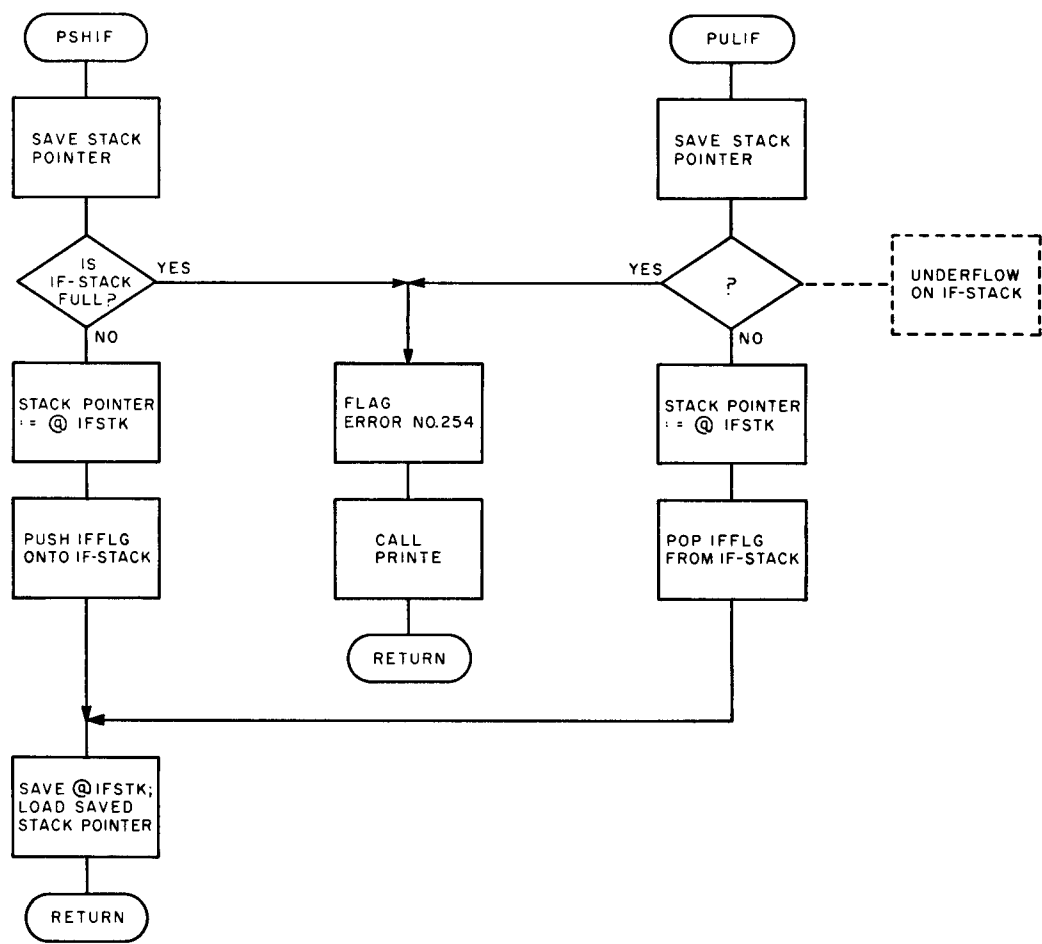
Flowchart of POIF routine.

PSHIF (PULIF)

This routine is used by the POIF and PONIF routines to either push or pull the present value of the IFFLG on (from) the IF stack (IFSTK).

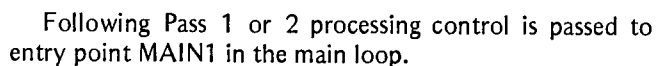
Calls: PRINTE
Called By: POIF, PONIF
Entry Points: PULIF
Flags: IFFLG
Pointers: STKSAV, @IFSTK

Flowchart of PUSHIF and PULIF routines.



This routine processes the MAC pseudo operation. During Pass 1 POMAC stores in the Macro Table (MACTBL) all of the source lines following the MAC pseudo instruction up to, but not including, the statement containing the MEND pseudo instruction. The Macro name is stored in the Symbol Table and the value of this name is the starting address in the MACTBL where the Macro definition is stored. The flag byte in the Symbol Table entry is set to indicate that the entry is a Macro name.

Flowchart of POMAC routine.



Calls: ADRINT, COMPAR, MACMOV, NXTOK,
PRINTE, PRINTL, RDLIN

Called By: MAIN

Flags: CMNFLG, LBFLG, MACERR, PASS

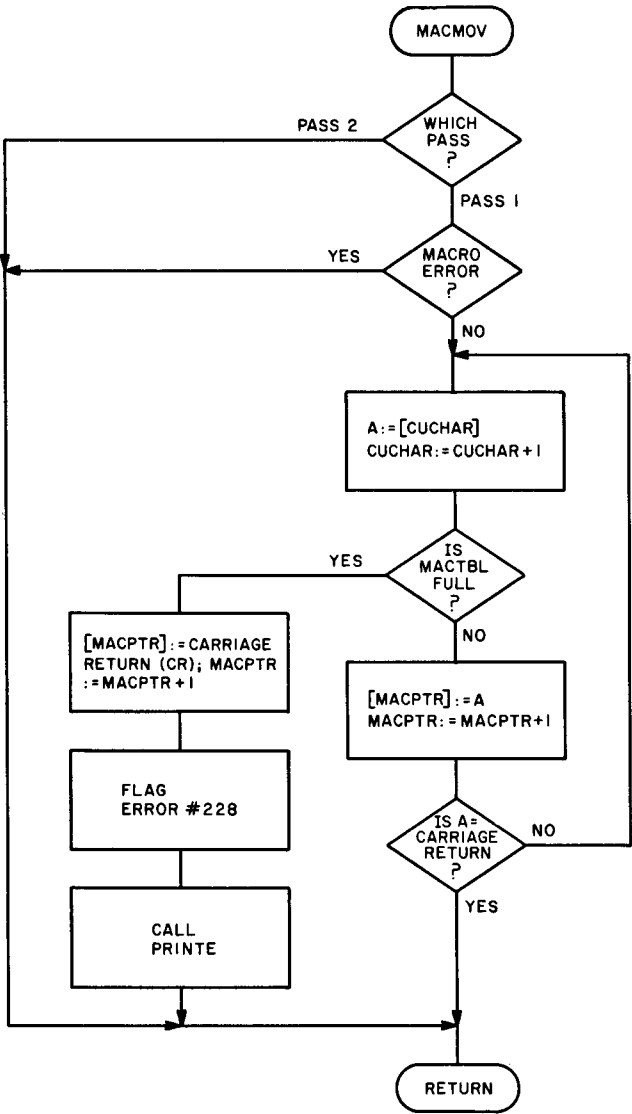
Pointers: CULINE, DESCRA, LNUM, MACPTR,
PCOUNT, SYMPTR

MACMOV

This routine is used by the POMAC routine to move a line from the Macro definition to the Macro Table.

Calls: PRINTE
Called By: POMAC
Flags: MACERR, PASS
Pointers: CULINE, MACPTR

Flowchart of MACMOV routine. The use of square brackets as in [Y] indicates the contents at address Y.

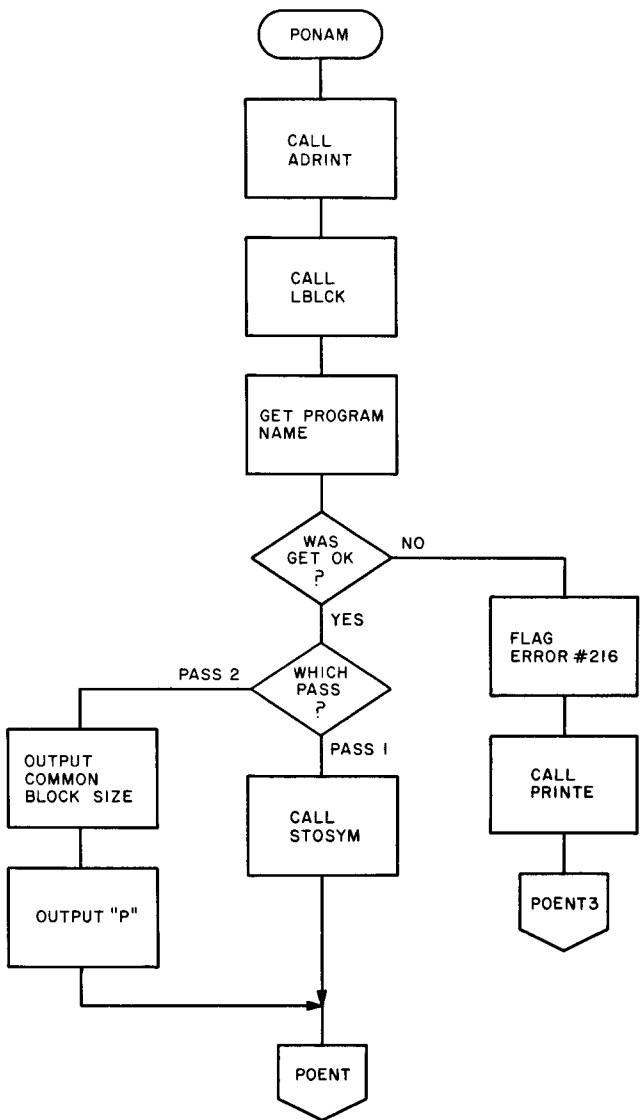


PONAM

This routine processes the NAM pseudo instruction. The operand name is passed to the Linking Loader as an Entry point followed by the size of the Common Block.

Calls: ADRINT, LBLCK, NXTOK, OUTBIN, OUTBNR, PRINTE, PRINTL
Jumps: POENT1, POENT3
Called By: MAIN
Flags: PASS
Pointers: CMNLC

Flowchart of PONAM routine.

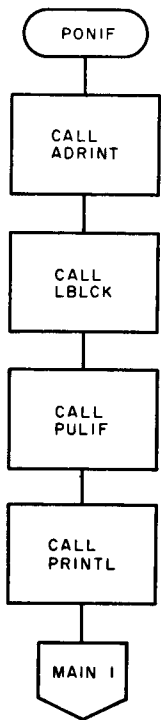


PONIF

This routine processes the NIF pseudo operation. There is no distinction between Pass 1 and Pass 2 processing.

The last value of the flag IFFLG is pulled off of the IFSTACK.

Calls: ADRINT, LBLCK, PRINTE, PULIF
Jumps: MAIN1
Called By: MAIN



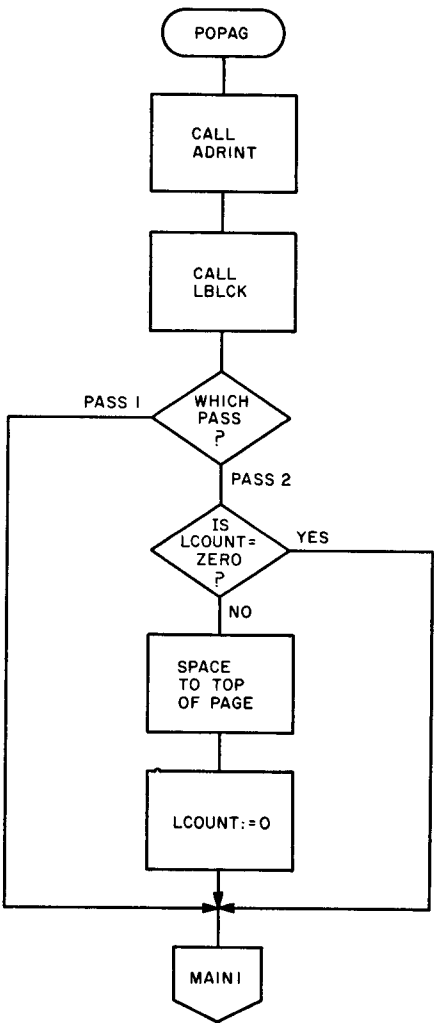
Flowchart of PONIF routine.

POPAG

This routine processes the PAG pseudo instruction. During Pass 2 this routine simply advances the listing on the system console (if the listing option has been selected) to the top of the next page.

Following Pass 1 or 2 processing control is passed to entry point MAIN1 in the main loop.

Calls: ADRINT, LBLCK, OUTCHR
Called By: MAIN
Flags: PASS
Pointers: LCOUNT



Flowchart of POPAG routine.

Flowchart of PORMB routine. The use of square brackets as in [Y] indicates the contents at address Y.

PORMB

This routine processes the RMB pseudo instruction. During Pass 1 the operand is evaluated and the Location Counter (LC) is incremented by this value. In Pass 2 the Location Counter is incremented by the value of the operand, and that number of zeroes is stored in the output buffer.

Following Pass 1 or 2 processing control is passed to entry point MAIN1 in the main loop.

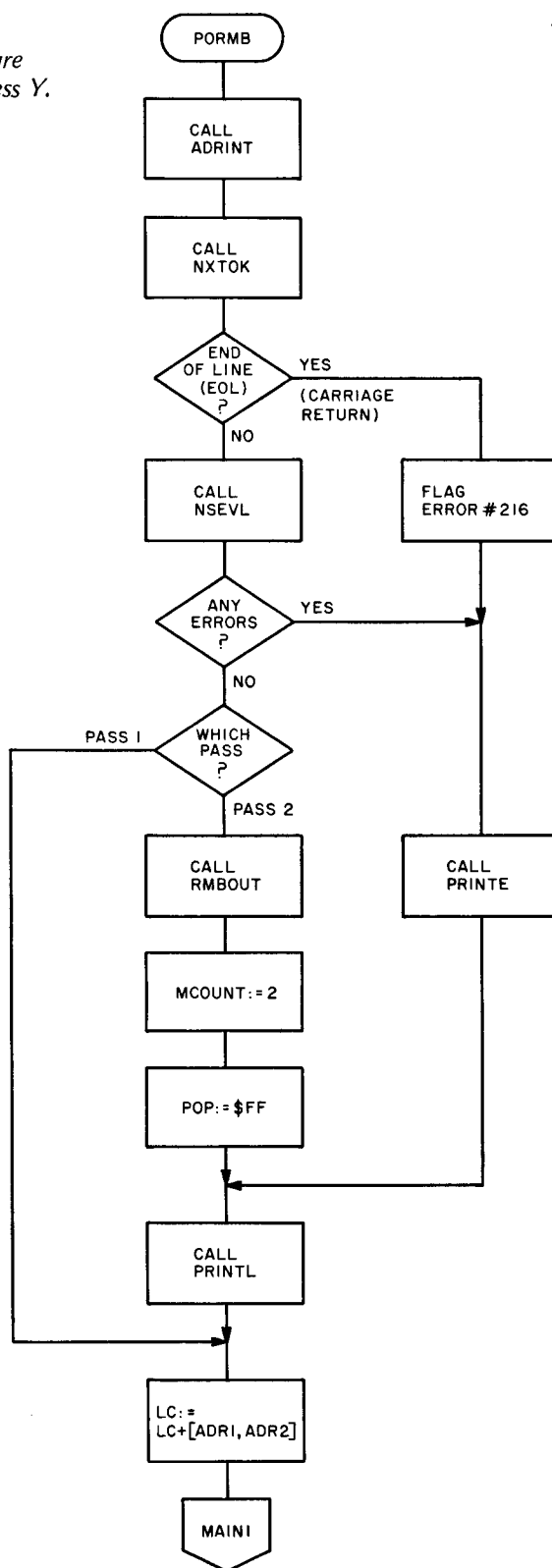
Calls: ADRINT, NSEVL, NXTOK, PRINTE, PRINTL, RMBOUT

Called By: MAIN

Flags: MCOUNT, PASS, POP

Pointers: ADR1, ADR2, LC

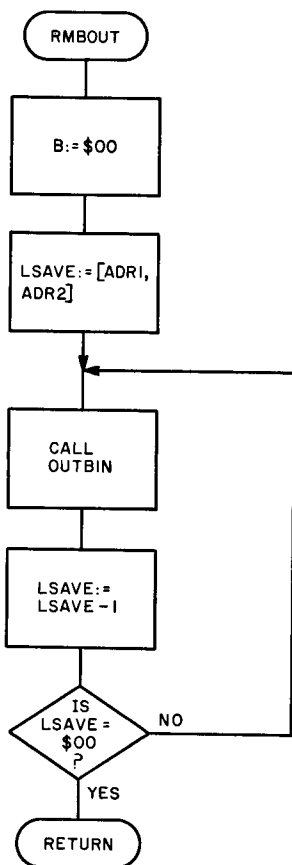
Temporaries: LSAVE



RMBOUT

This routine is used by the PORMB routine to output zero bytes to the output file. On entry, (ADR1, ADR2) contains the number of bytes to be output.

Calls: OUTBIN
 Called By: PORMB
 Pointers: ADR1, ADR2
 Temporaries: LSAVE

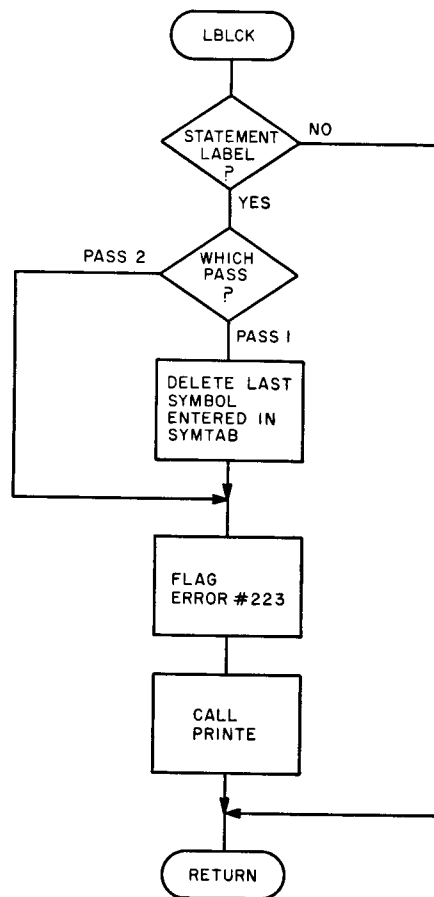


Flowchart of RMBOUT routine. The use of square brackets as in [Y] indicates the contents at address Y.

LBLCK

This routine is used by certain of the pseudo instruction processing routines to check to see if there is a label for that source line. If there is a label, it is deleted from the Symbol Table, and an error message printed.

Calls: DELSYM, PRINTE
 Called By: POCMN, POEND, POENT, POEXT, PONAM, POPAG
 Flags: LBFLG, PASS



Flowchart of LBLCK routine.

Opcode Processing Routines

The address processing routines all perform the same function and share common subroutines. The main function of an address processing routine is to scan the *operand* field of a statement and, based on the structure and values of the operands, generate the machine code for the instruction being processed. The processing routine also increments the Location Counter by the number of bytes the assembled instruction requires. Inherent, Relative and Accumulator addressing types require one byte. Indexed and Direct types require two bytes. Extended type instructions require three bytes, and Immediate types require either two or three bytes.

During Pass 1 no machine code is generated, but the *operand* field is scanned to detect errors and to calculate the number of bytes the instruction requires.

During Pass 2 the machine code is generated and stored in the output file.

Following Pass 1 and 2 processing control is passed to entry point MAIN1 in the main loop.

ADDR1

This routine processes the following opcodes: ADC, ADD, AND, BIT, CMP, LDA, ORA, SBC, SUB. The operand structure may be Immediate (2 bytes), Direct (2 bytes), Extended (3 bytes), or Indexed (2 bytes).

On entry, register B contains part of the opcode: depending on the *operand* field, ADDR1 completes the opcode, generates the machine code for the address part of the instruction, and outputs it to the output file in Pass 2.

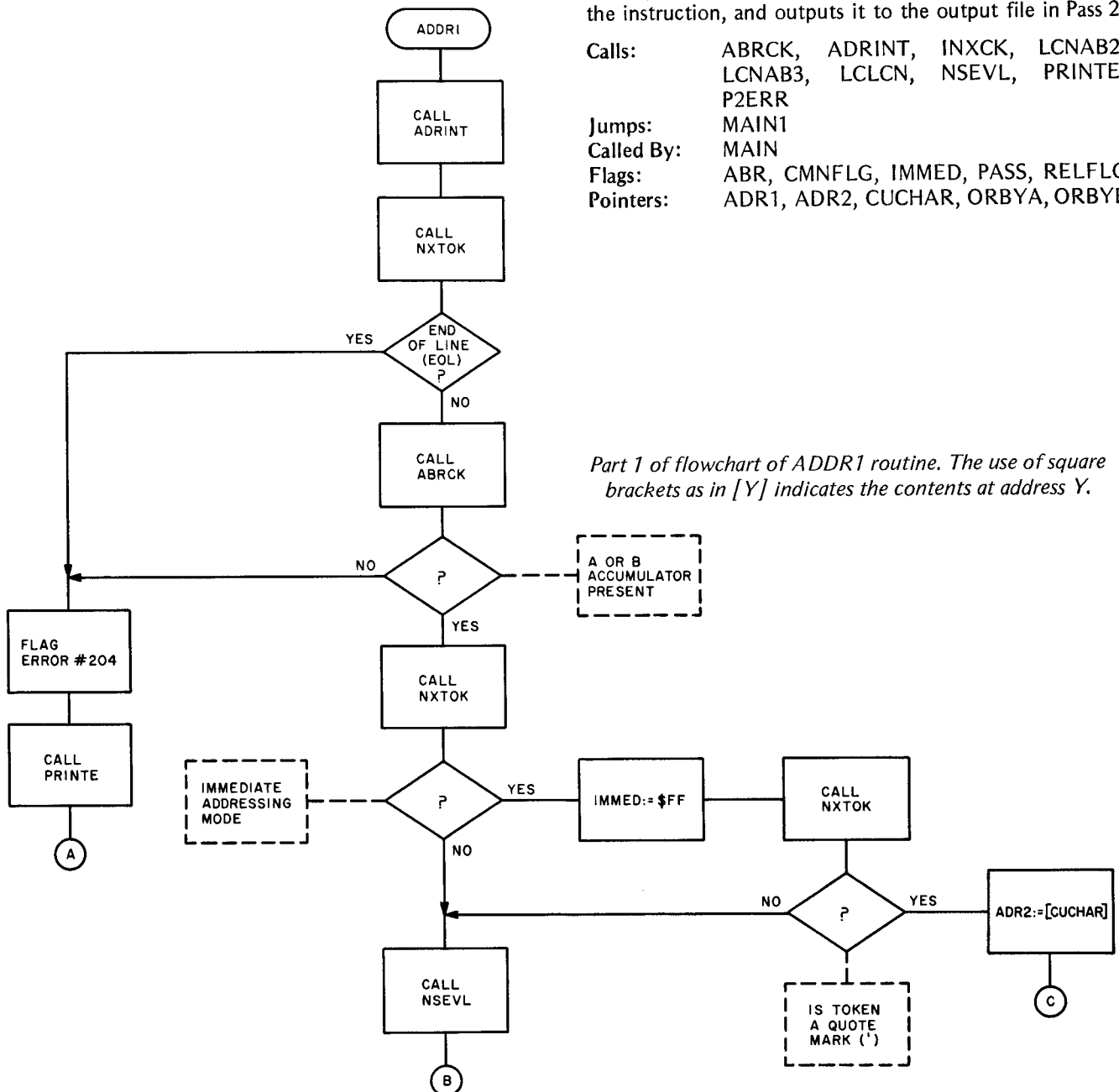
Calls: ABRCK, ADRINT, INXCK, LCNAB2, LCNAB3, LCLCN, NSEVL, PRINTE, P2ERR

Jumps: MAIN1

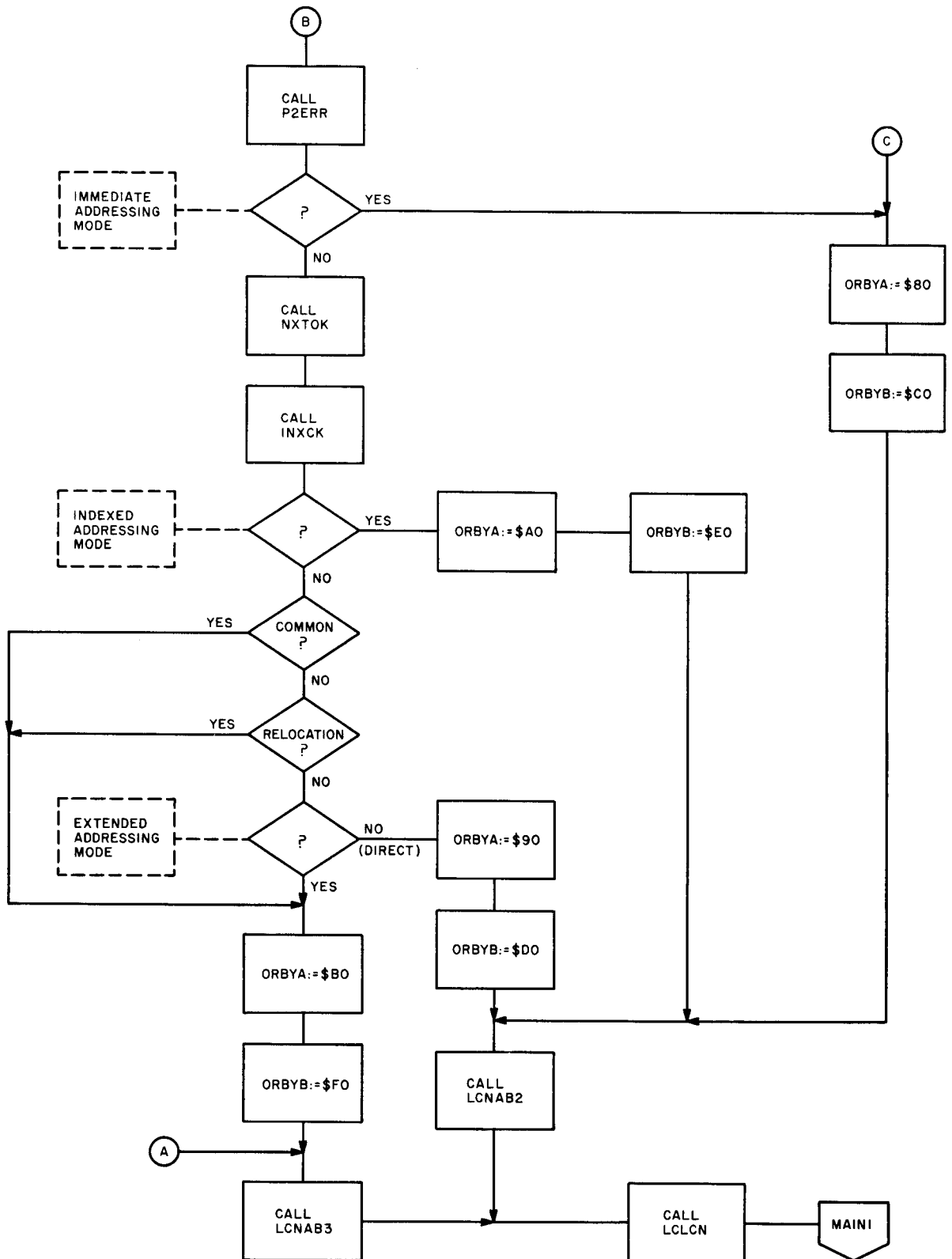
Called By: MAIN

Flags: ABR, CMNFLG, IMMED, PASS, RELFLG

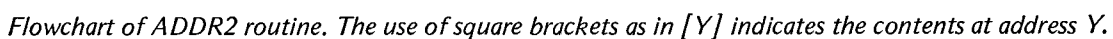
Pointers: ADR1, ADR2, CUCHAR, ORBYA, ORBYB



Part 2 of flowchart of ADDR1 routine.



Pointers: ADR1, ORBYA, ORBYB



ADDR3

This routine processes the following opcodes: ASL, ASR, CLR, COM, DEC, INC, LSR, NEG, ROL, ROR, TST. The operand structure may be Accumulator (1 byte), Extended (3 bytes), or Indexed (2 bytes).

On entry, register B contains part of the opcode; depending on the *operand* field, ADDR3 completes the opcode, generates the machine code for the address field of the instruction, and sends it to the output file.

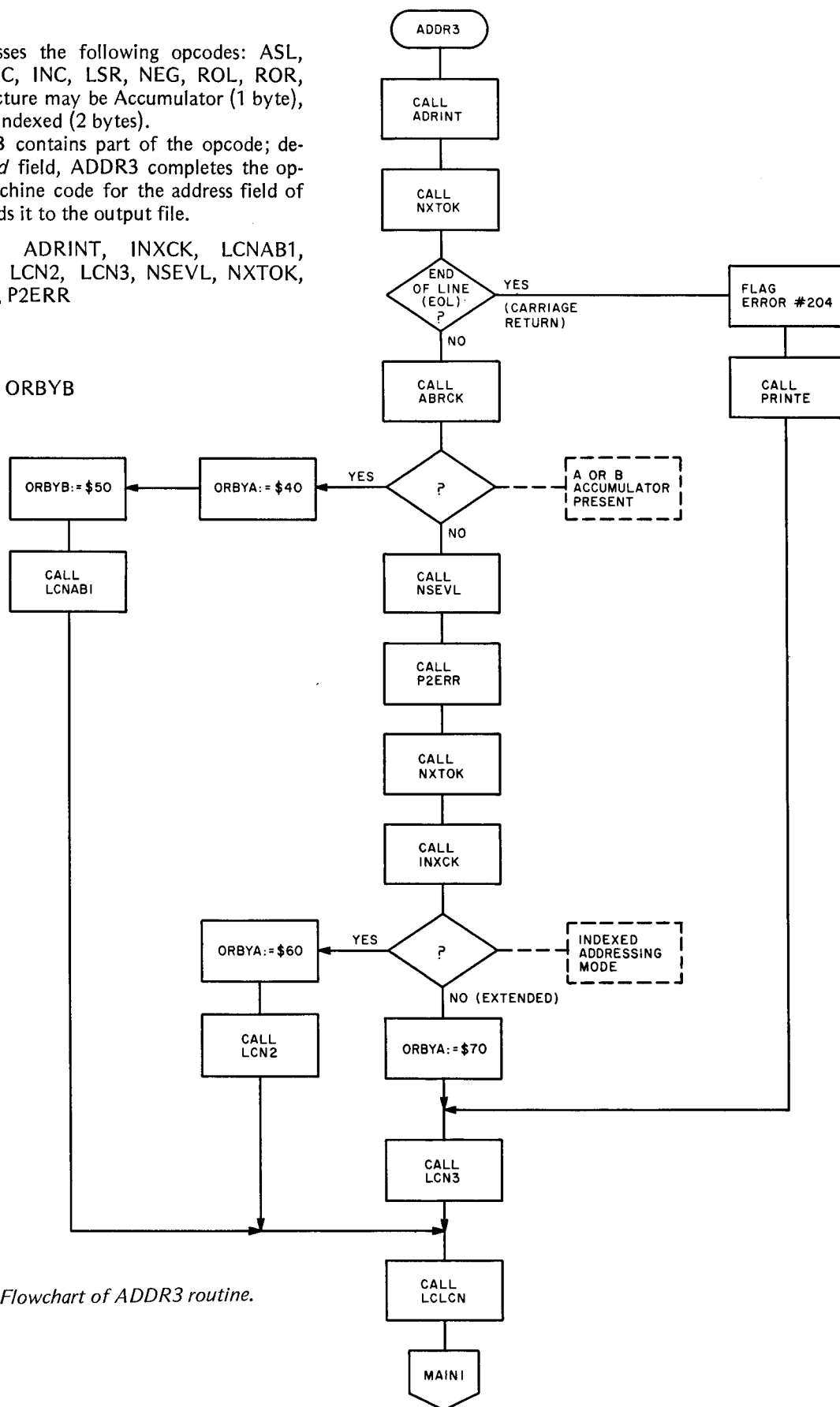
Calls: ABRCK, ADRINT, INXCK, LCNAB1, LCLCN, LCN2, LCN3, NSEVL, NXTOK, PRINTE, P2ERR

Jumps: MAIN1

Called By: MAIN

Flags: ABR

Pointers: ORBYA, ORBYB

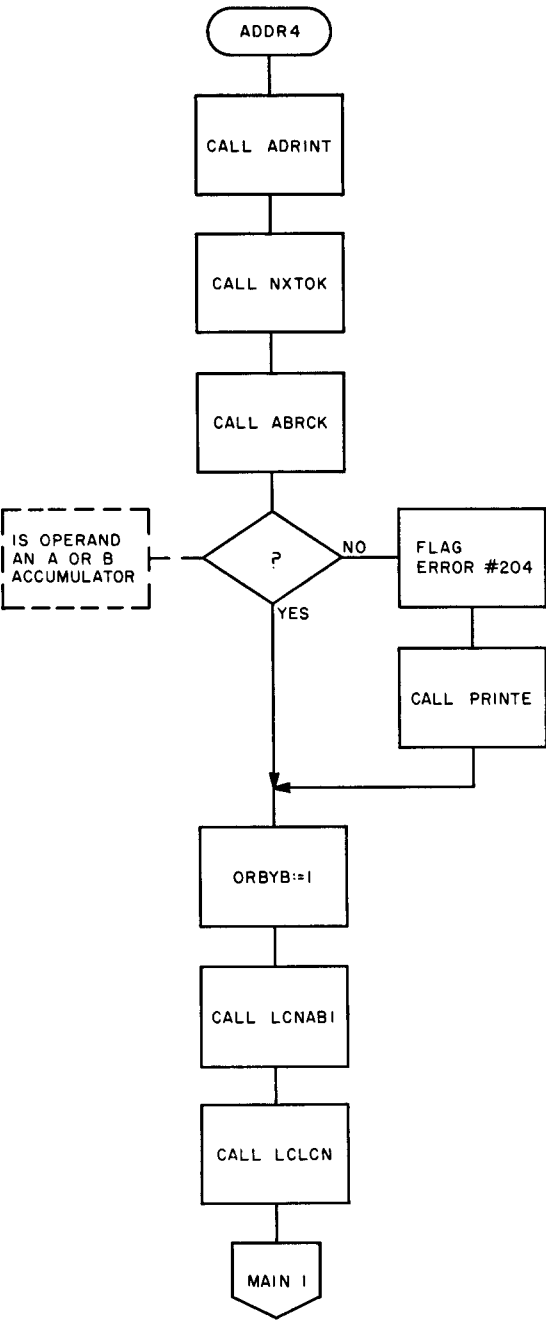


Flowchart of ADDR3 routine.

ADDR4

This routine processes the opcodes PSH and PUL. The operand structure is the Accumulator structure (1 byte).
On entry, register B contains the partial opcode. Depending on the Accumulator in the *operand* field, ADDR4 completes the opcode and outputs it to the output file in Pass 2.

Calls: ABRCK, ADRINT, LCLCN, LCNAB1, NXTOK, PRINTE
Jumps: MAIN1
Called By: MAIN
Flags: ABR
Pointers: ORBYA, ORBYB



Flowchart of ADDR4 routine.

ADDR5

This routine processes the following opcodes: CPX, LDS, LDX.

The operand structure may be Immediate (3 bytes), Direct (2 bytes), Extended (3 bytes), or Indexed (2 bytes).

On entry, register B contains the partial opcode. Depending on the *operand* field, ADDR5 completes the opcode, generates the machine code for the address part of the instruction, and outputs it to the output file in Pass 2.

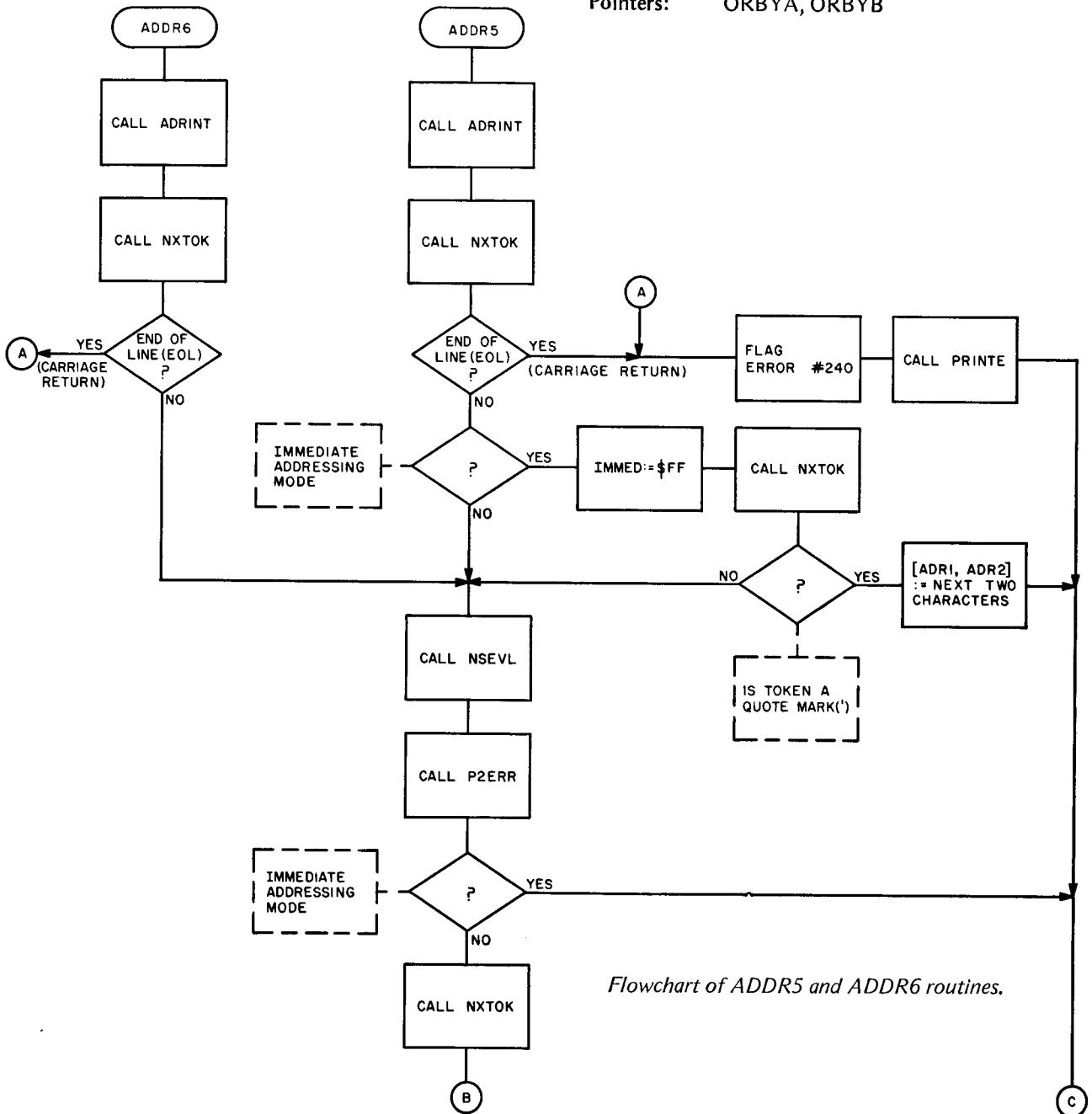
Calls: ADRINT, INXCK, LCLCN, LCN2, LCN3, NSEVL, NXTOK, PRINTE, P2ERR

Jumps: MAIN1

Called By: MAIN

Flags: CMNFLG, RELFLG

Pointers: ORBYA, ORBYB



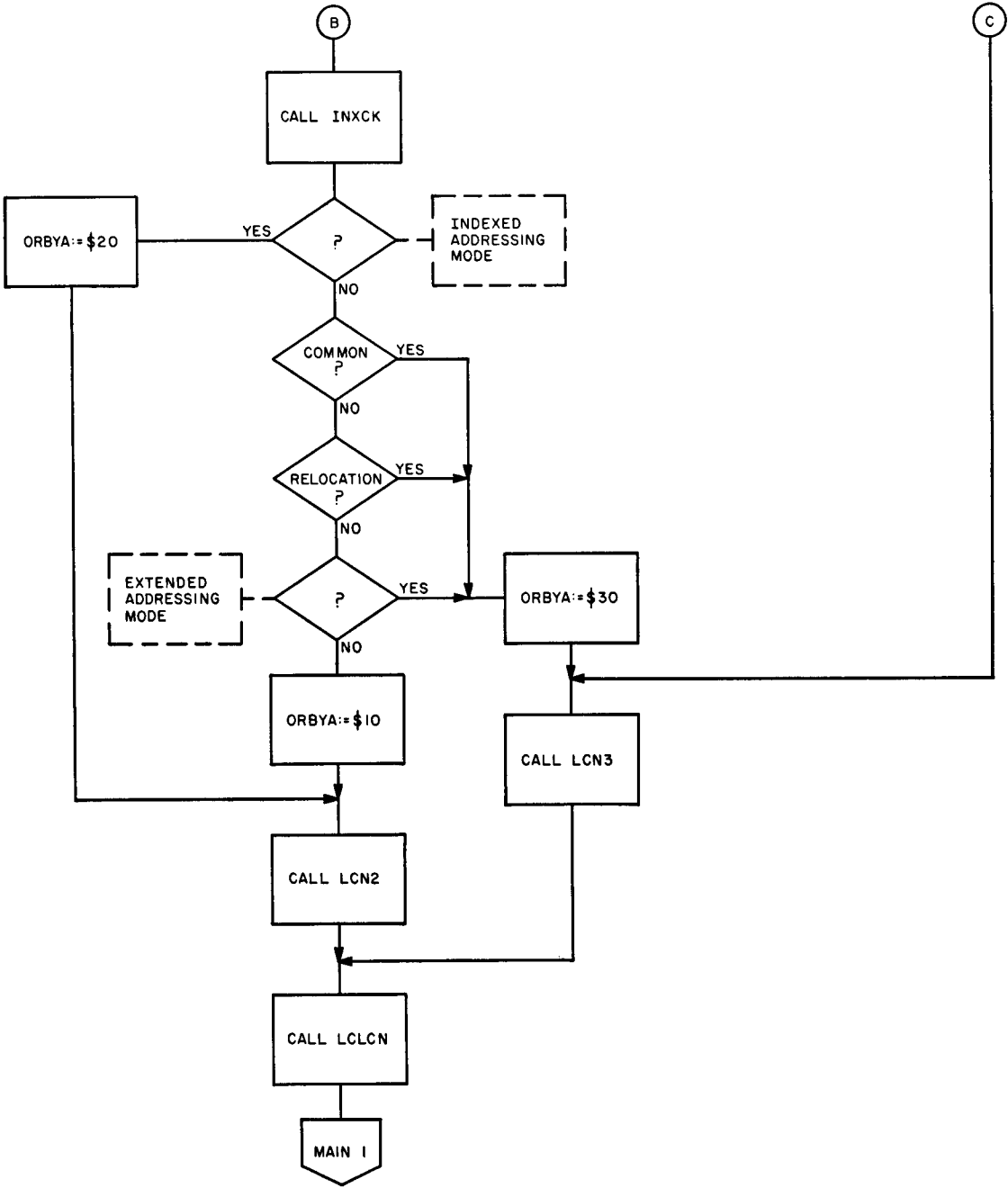
Flowchart of ADDR5 and ADDR6 routines.

ADDR6

This routine processes the following opcodes: STX, STS. The operand structure may be Direct (2 bytes), Extended (3 bytes), or Indexed (2 bytes).

On entry, register B contains the partial opcode. Depending on the *operand* field, ADDR6 completes the opcode, generates the machine code for the address part of the instruction, and outputs it to the output file in Pass 2.

Calls: ADRINT, NXTOK
Jumps: ADDR5A, ADDR5C
Called By: MAIN



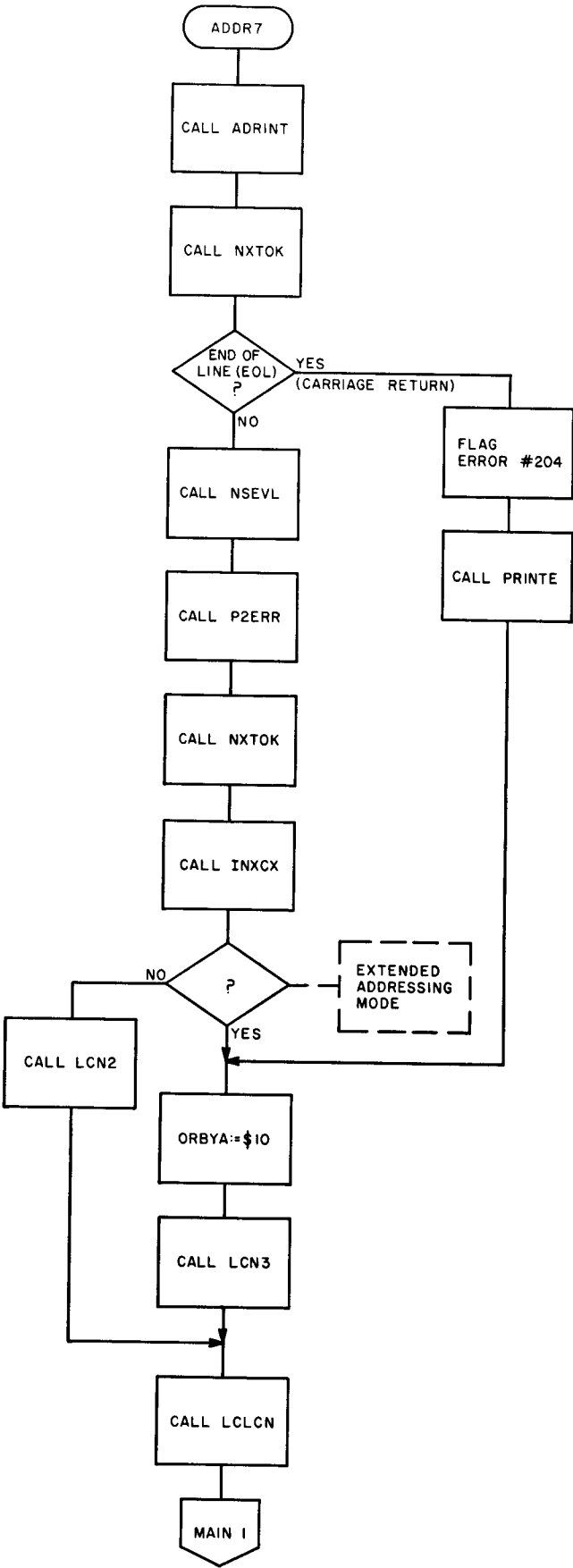
ADDR7

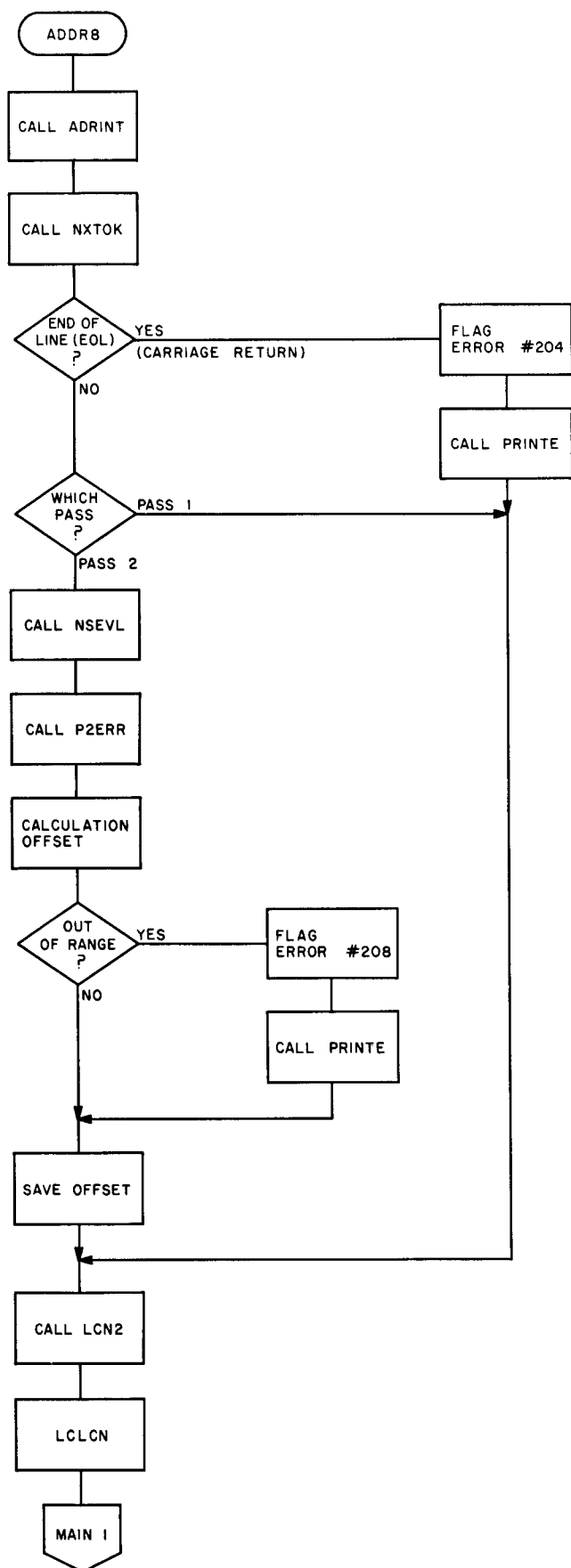
This routine processes the following opcodes: JMP, JSR. The operand structure may be Extended (3 bytes), or Indexed (2 bytes).

On entry, register B contains the partial opcode. Depending on the *operand* field, ADDR7 completes the opcode, generates the machine code for the address part of the instruction, and outputs it to the output file in Pass 2.

- Calls:
- ADRIINT, INXCK, LCLCN, LCN2, LCN3, NSEVL, NXTOK, PRINTE, P2ERR
- Jumps:
- MAIN1
- Called By:
- MAIN
- Pointers:
- ORBYA

Flowchart of ADDR7 routine.





ADDR8

This routine processes the following opcodes: BCC, BCS, BEQ, BGE, BGT, BHI, BLE, BLT, BMI, BRA, BSR, BVC, BVS. The operand structure is the Relative (2 bytes).

On entry, register B contains the complete opcode. ADDR8 evaluates the *operand* field and calculates the relative offset that is to be the address part of the instruction and outputs it to the output buffer in Pass 2.

Calls: ADRINT, LCLCN, LCN2, NSEVL, NXTOK, PRINTE, P2ERR

Jumps: MAIN1

Called By: MAIN

Flags: PASS

Pointers: ADR1, ADR2, LC

Temporaries: LSAVE

Flowchart of ADDR8 routine.

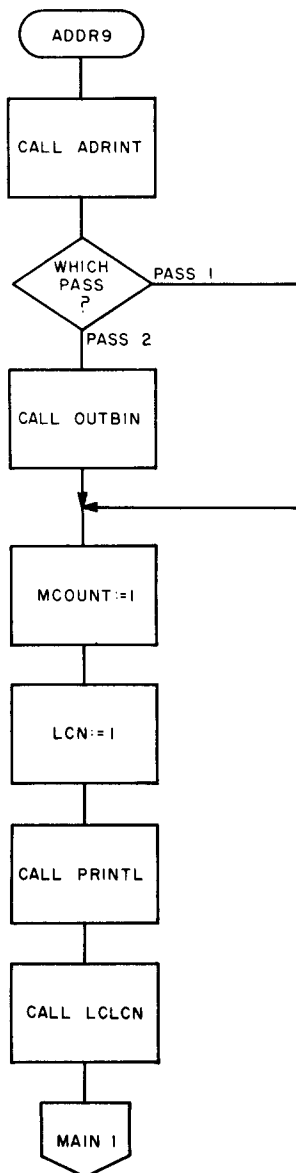
ADDR9

This routine processes the following opcodes: ABA, CBA, CLC, CLI, CLV, DAA, DES, DEX, INS, INX, NOP, RTI, RTS, SBA, SEC, SEI, SEV, SWI, TAB, TAP, TBA, TPA, TSX, TXS, WAI.

The operand structure does not exist as this is an Inherent type instruction. On entry register B contains the complete opcode, and the routine outputs this value to the output buffer in Pass 2.

Calls: ADRINT, LCLCN, OUTBIN, PRINTL
Jumps: MAIN1
Called By: MAIN
Flags: MCOUNT, PASS
Pointers: LCN

Flowchart of ADDR9 routine.



Address Processing Utility Routines

These utility routines are used by the opcode processing routines ADDR1 through ADDR9 for processing the various operands and instruction types.

ADRINT

This initializes flags and variables used in the opcode and pseudo operation processing routines.

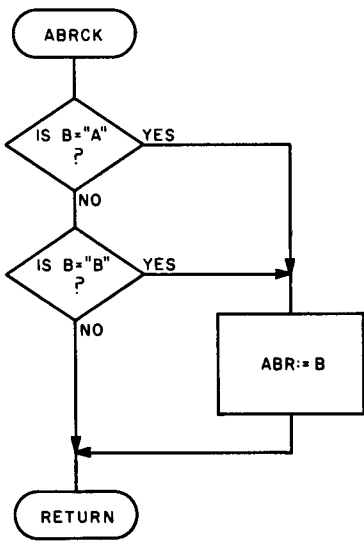
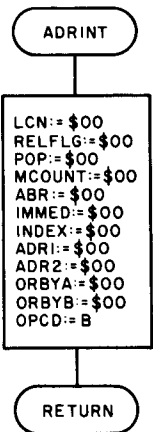
Calls: none
Called By: ADDR1 thru ADDR9, MAIN, POCMN, POEND, POENT, POEQU, POEXT, POFCB, POFCC, POFDB, POIF, POMAC, PONAM, PONIF, POPAG, PORMB
Flags: ABR, ADR1, ADR2, CMNFLG, ENTFLG, EXTFLG, IMMED, INDEX, LCN, MCOUNT, ORBYA, ORBYB, POP, RELFLG
Pointers: OPCD

ABRCK

This checks to see what, if any, register is the first operand in the *operand* field of an instruction. The register is either A or B.

Calls: none
Called By: ADDR1 thru ADDR4
Flags: ABR

Flowchart of ADRINT routine.



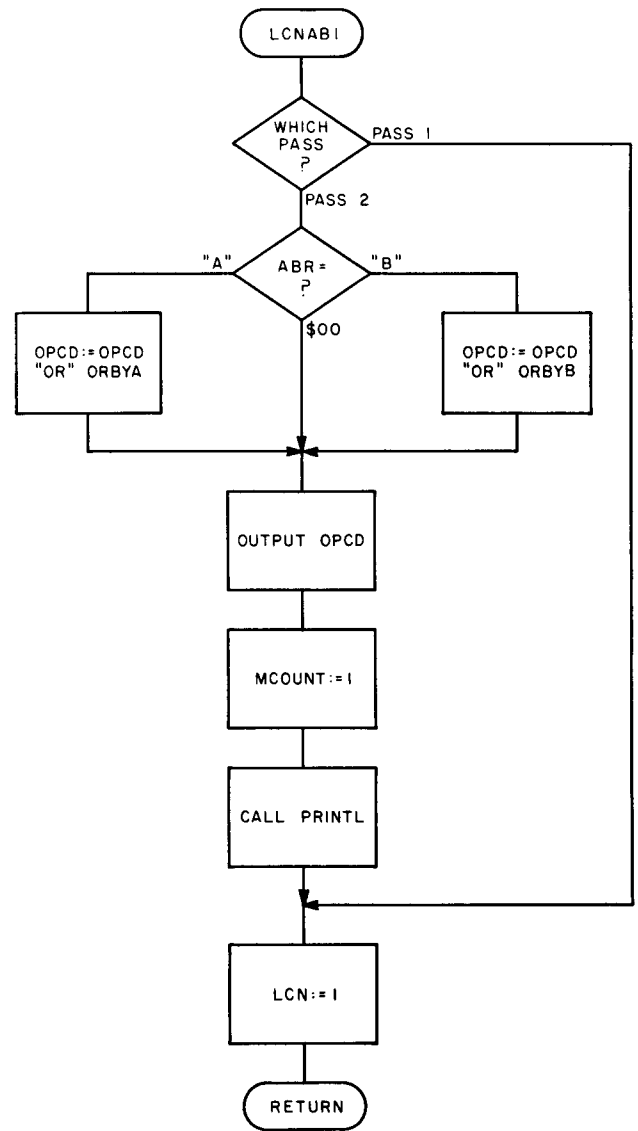
Flowchart of ABRCK routine.

LCNAB1

This does the finish up processing for one byte Accumulator type instructions.

Calls: OUTBIN, PRINTL
Called By: ADDR3, ADDR4
Flags: ABR, MCOUNT, PASS
Pointers: LCN, OPCODE, ORBYA, ORBYB

Flowchart of LCNAB1 routine.

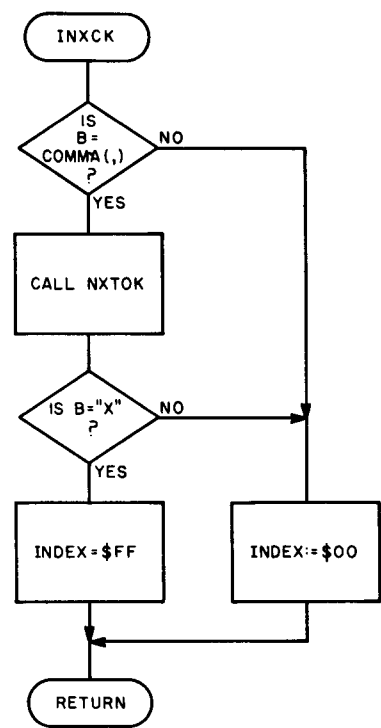


INXCK

This checks to see if an instruction is Indexed.

Calls: NXTOK, PRINTE
Called By: ADDR1, ADDR2, ADDR3, ADDR5, ADDR7
Flags: INDEX

Flowchart of INXCK routine.

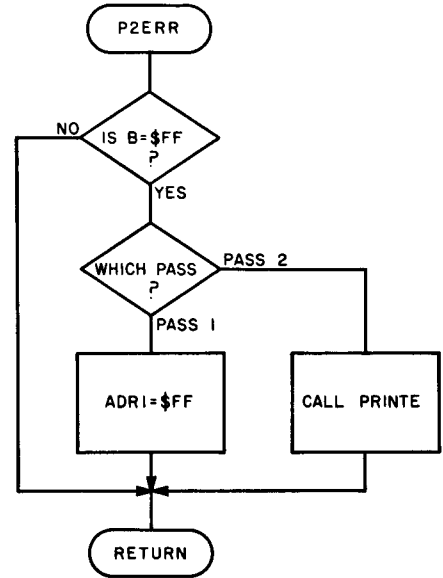


P2ERR

This prints Pass 2 errors. Some errors returned by the evaluation routine NSEVL are considered errors in Pass 2 but are not errors in Pass 1.

Calls: PRINTE
Called By: ADDR1, ADDR2, ADDR3, ADDR5, ADDR7, ADDR8, POFCB, POFDB
Flags: PASS
Pointers: ADR1, ADR2

Flowchart of P2ERR routine.



LCN2

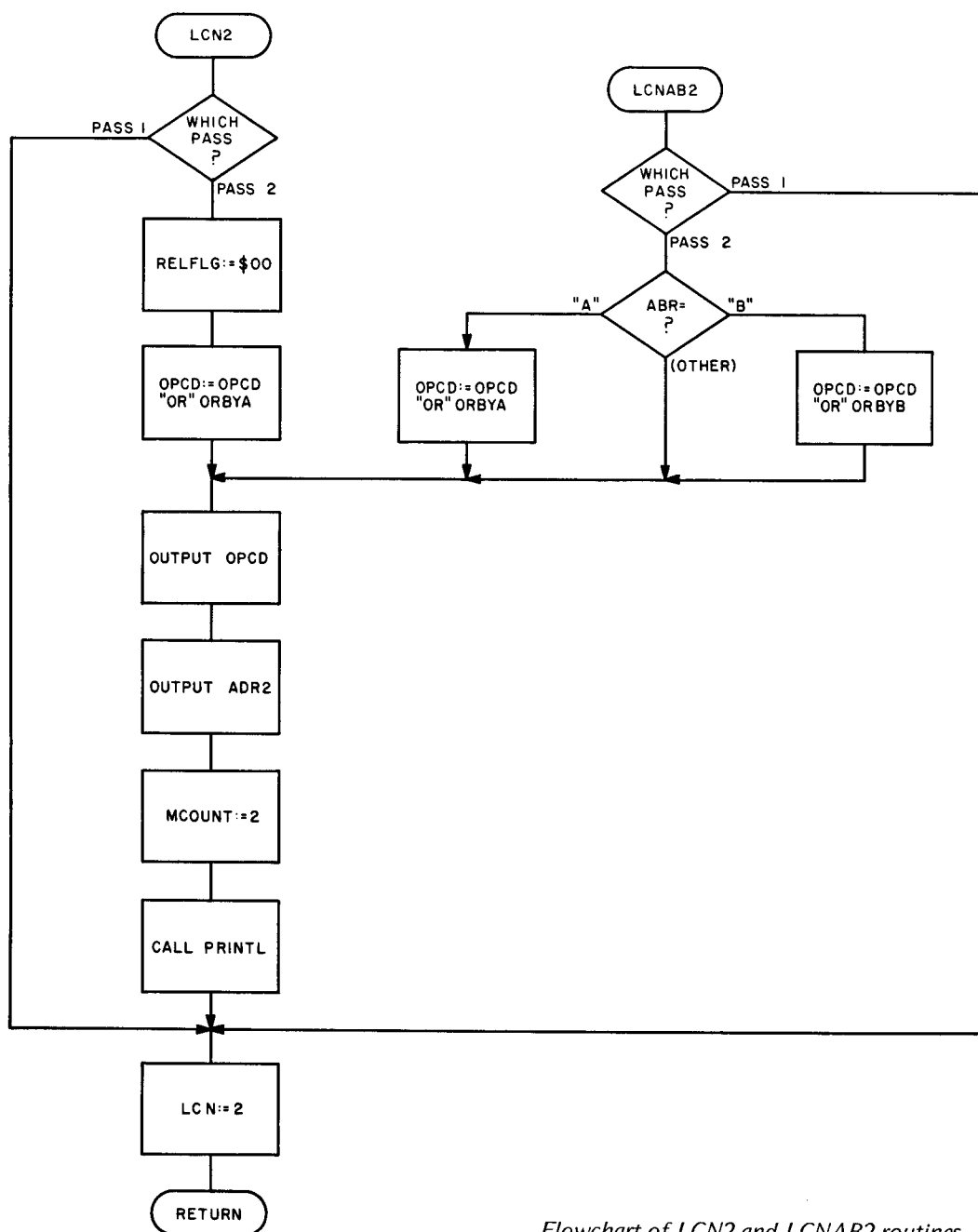
This does the finish up processing for two byte Indexed, Direct, and Immediate type instructions.

Calls: OUTBIN, PRINTL
Called By: ADDR3, ADDR5, ADDR7, ADDR8
Entry: LCN2A
Flags: CMNFLG, MCOUNT, PASS, RELFLG
Pointers: ADR2, LCN, OPCODE, ORBYA, ORBYB

LCNAB2

This does the finish up processing for two byte register (A, B) Indexed, Direct, and Immediate type instructions.

Calls: none
Jumps: LCN2A
Called By: ADDR1, ADDR2
Flags: ABR, PASS
Pointers: OPCODE, ORBYA, ORBYB



Flowchart of LCN2 and LCNAB2 routines.

LCN3

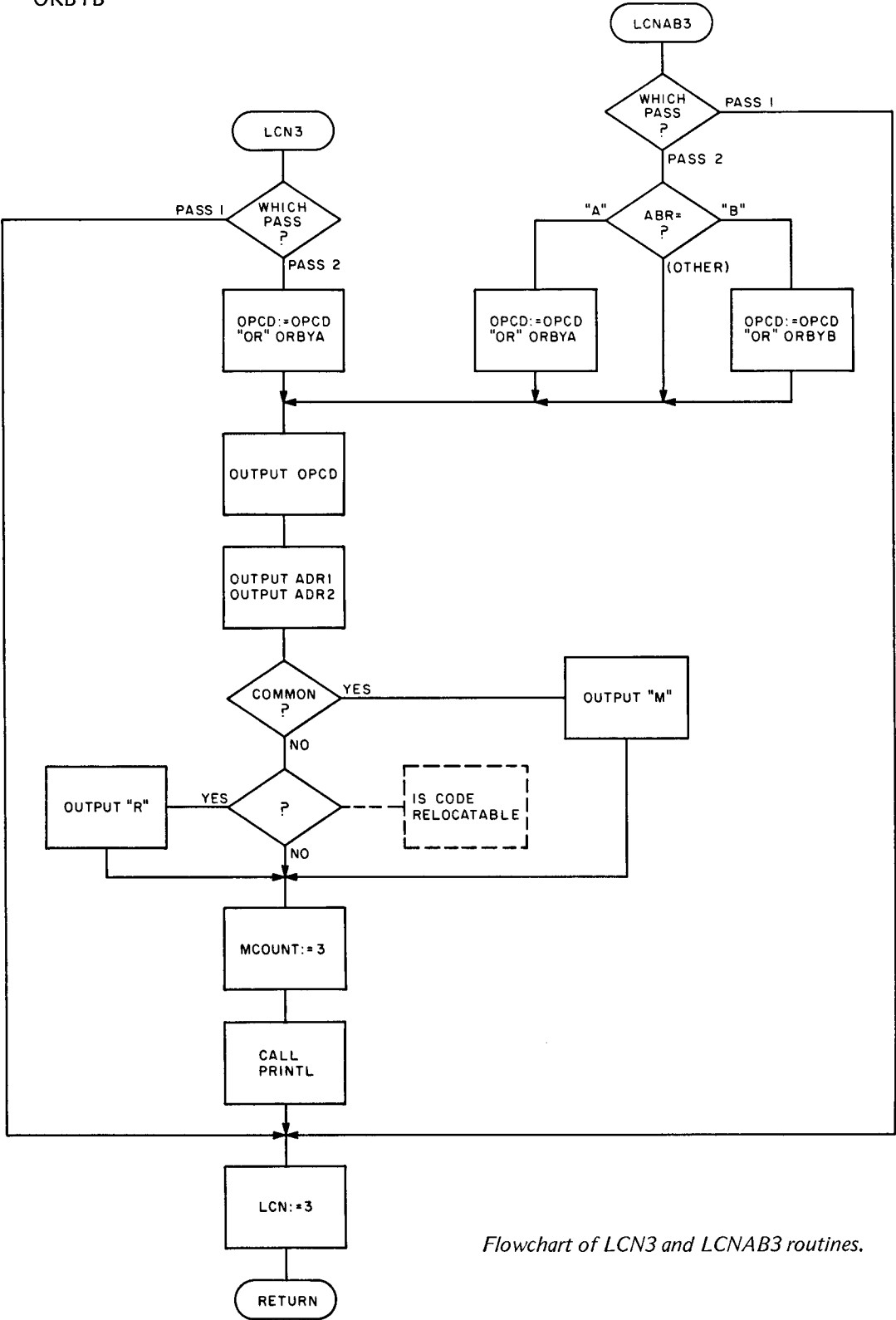
This does the finish up processing for the three byte Extended type instructions.

Calls: OUTBIN, PRINTL
Called By: ADDR3, ADDR5, ADDR7
Entry: LCN3A
Flags: CMNFLG, MCOUNT, PASS, RELFLG
Pointers: ADR1, ADR2, LCN, OPCODE, ORBYA, ORBYB

LCNAB3

This does the finish up processing for the three byte register (A, B) Extended and Immediate type instructions.

Calls: none
Jumps: LCN3A
Called By: ADDR1, ADDR2
Flags: ABR, PASS
Pointers: OPCODE, ORBYA, ORBYB



Flowchart of LCN3 and LCNAB3 routines.

LCLCN

This does the addition of LCN to LC (LC:=LC+LCN).
Calls: none
Called By: POEXT, POFCB, POFCC, POFDB, ADDR1 thru ADDR5, ADDR7 thru ADDR9
Pointers: LC, LCN

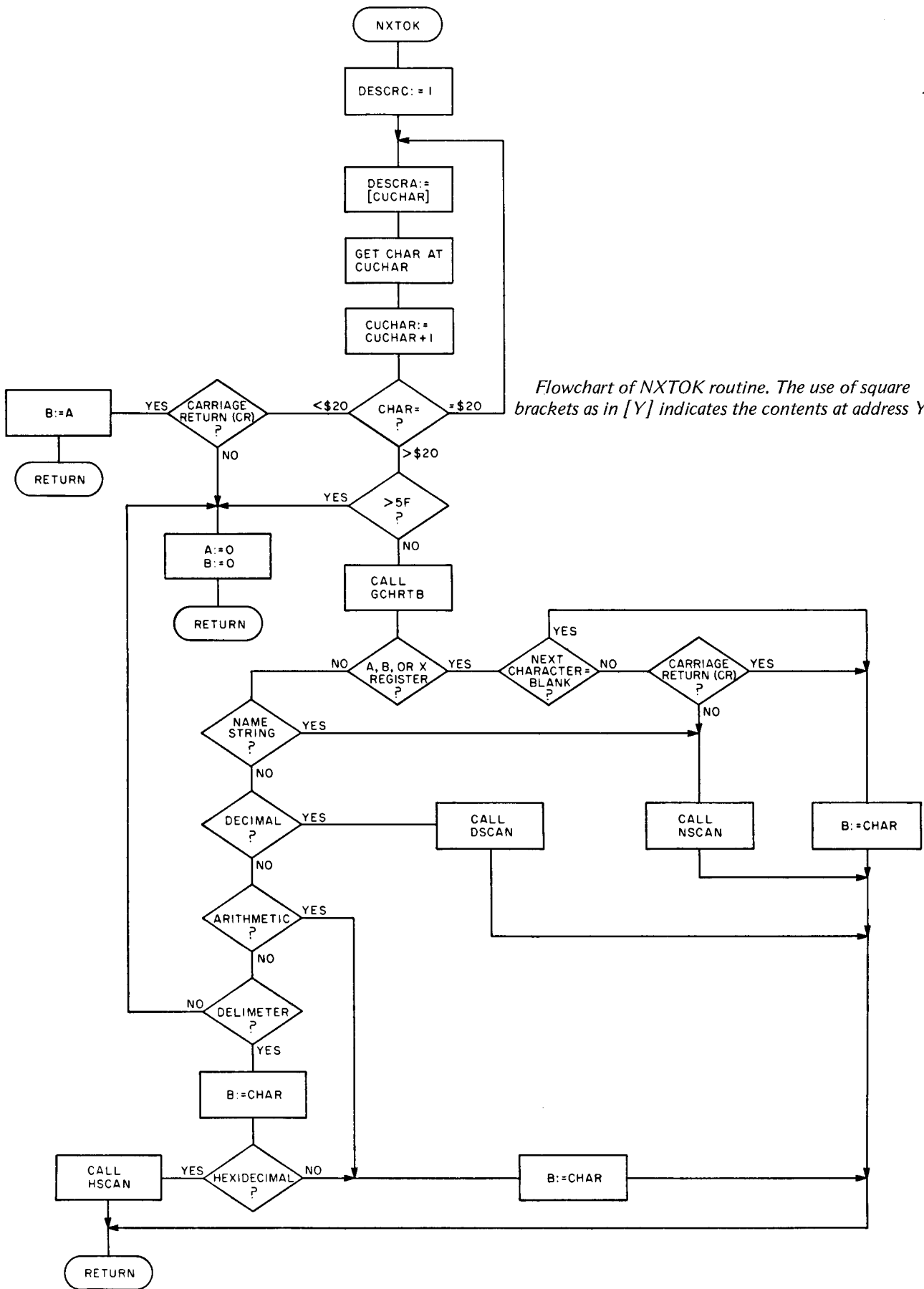
Lexical Analysis Routines

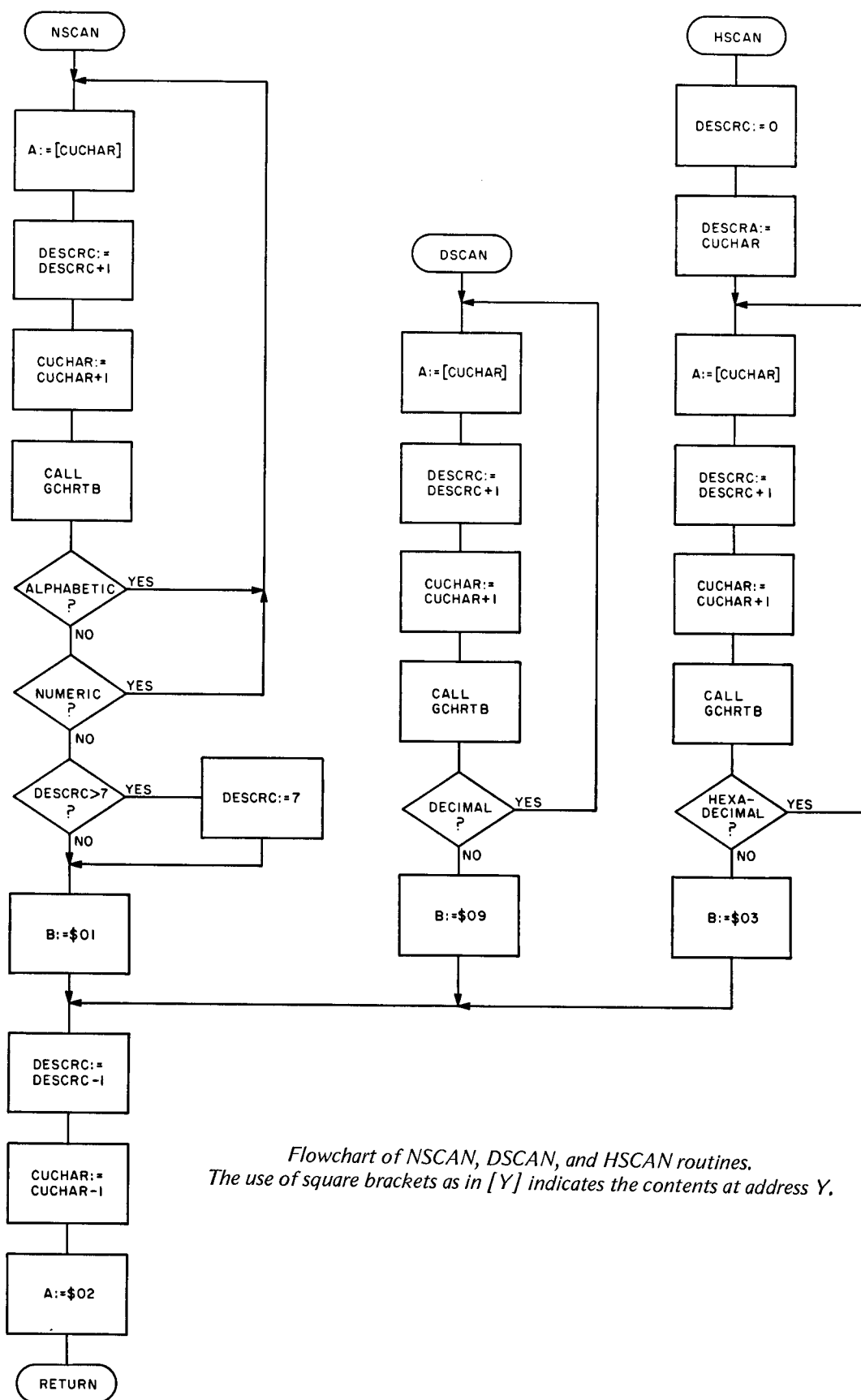
The lexical analysis routines described in this section are concerned with finding and classifying the individual tokens of an assembly language statement. A token is a non-blank string of contiguous characters, such as a label, an expression, or an operand.

NXTOK

This routine extracts tokens from a line of source code. It scans a line of source code and returns the next token each time that it is called.
On entry, CUCHAR points to the next character in the line. NXTOK returns a token by placing the address of the token in DESCRA and the length of the token in DESCRC. The routine also returns the token type in register B and the token class in register A. If the token is unrecognizable, the routine returns with both the A and B registers cleared. The following tokens are recognized by NXTOK:

TOKEN	TYPE (B)	CLASS (A)	
NAME	01	02	Substrings
HEX	03	02	
DECIMAL	09	02	
#	23	04	Delimiters
,	2c	04	
,	27	04	
*	2A	24	Arithmetic
/	2F	24	
+	2B	24	
—	2D	24	
A	41	01	A,B,X registers
B	42	01	
X	58	01	
CR	0D	0D	End of Line
ERROR	00	00	Error
Calls:	DSCAN, GCHRTB, HSCAN, NSCAN		
Called By:	ADDR1 thru ADDR8, INXCK, MAIN, NSEVL, POCMN, POENT, POEQU, POEXT, POFCB, POFCC, POFDB, POMAC, PONAM, PORMB		
Pointers:	CUCHAR, DESCRA, DESCRC		





Flowchart of NSCAN, DSCAN, and HSCAN routines.
The use of square brackets as in [Y] indicates the contents at address Y.

DSCAN

This routine scans substrings of decimal characters. On entry, CUCHAR points to the first character to be scanned. DSCAN continues to scan until it finds a non-decimal character. The address of the decimal substring is returned in DESCRA and the length of the substring is returned in DESCRC. The B register is loaded with a type code of 09.

Calls: GCHRTB
Jumps: ENDSCN
Called By: NXTOK
Pointers: CUCHAR, DESCRC

NSCAN

This routine scans substrings of alphanumeric characters. On entry, CUCHAR points to the first character to be scanned. NSCAN continues to scan until it finds a non-alphanumeric character. The address of the alphanumeric substring is returned in DESCRA and the length of the substring is returned in DESCRC. The B register is loaded with a type code of 01.

Calls: GCHRTB
Jumps: ENDSCN
Called By: NXTOK
Pointers: CUCHAR, DESCRC

HSCAN

This routine scans substrings of hexadecimal characters. On entry, CUCHAR points to the first character to be scanned. HSCAN continues to scan until it finds a non-hexadecimal character. The address of the substring is returned in DESCRA and the length of the substring is returned in DESCRC. The B register is loaded with the type code 03.

Calls: GCHRTB
Called By: NXTOK
Pointers: CUCHAR, DESCRA, DESCRC

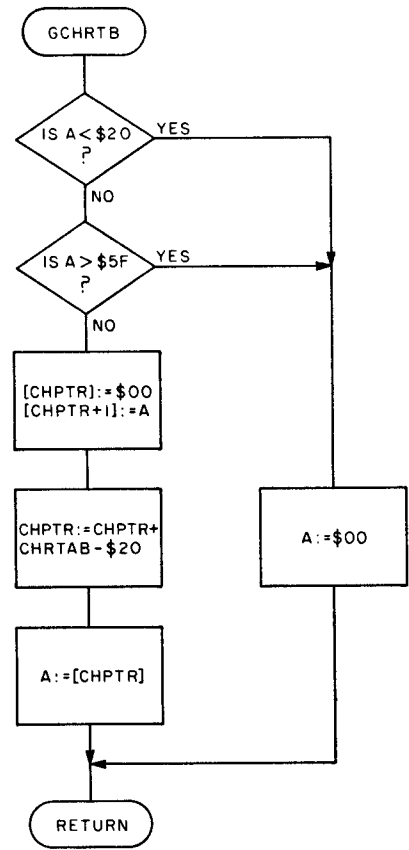
ENDSCN

This is a common return for routines: DSCAN, NSCAN, and HSCAN.

GCHRTB

This routine retrieves the byte in CHRTAB that is indexed by the value of the character in the A register. On return, register A contains the value of the byte retrieved from CHRTAB.

Calls: ADD16
Called By: DSCAN, HSCAN, NSCAN, NXTOK
Pointers: CHPTR



Flowchart of GCHRTB routine. The use of square brackets as in [Y] indicates the contents at address Y.

Evaluation Routine

NSEVL

This routine evaluates numbers, symbols, and expressions composed of numbers, symbols and operators. A straight left to right evaluation is performed without regard to precedence of hierarchy of operators.

The relocation indicator flag (RELFLG) is set if the final result is relocatable. Generally, a result is considered relocatable if it contains an odd count of relocatable terms. This can produce meaningless results; for example, the addition of two relocatable terms is an absolute value, but unfortunately not very useful. However, the difference of two relocatable terms can be very useful as the length of a table.

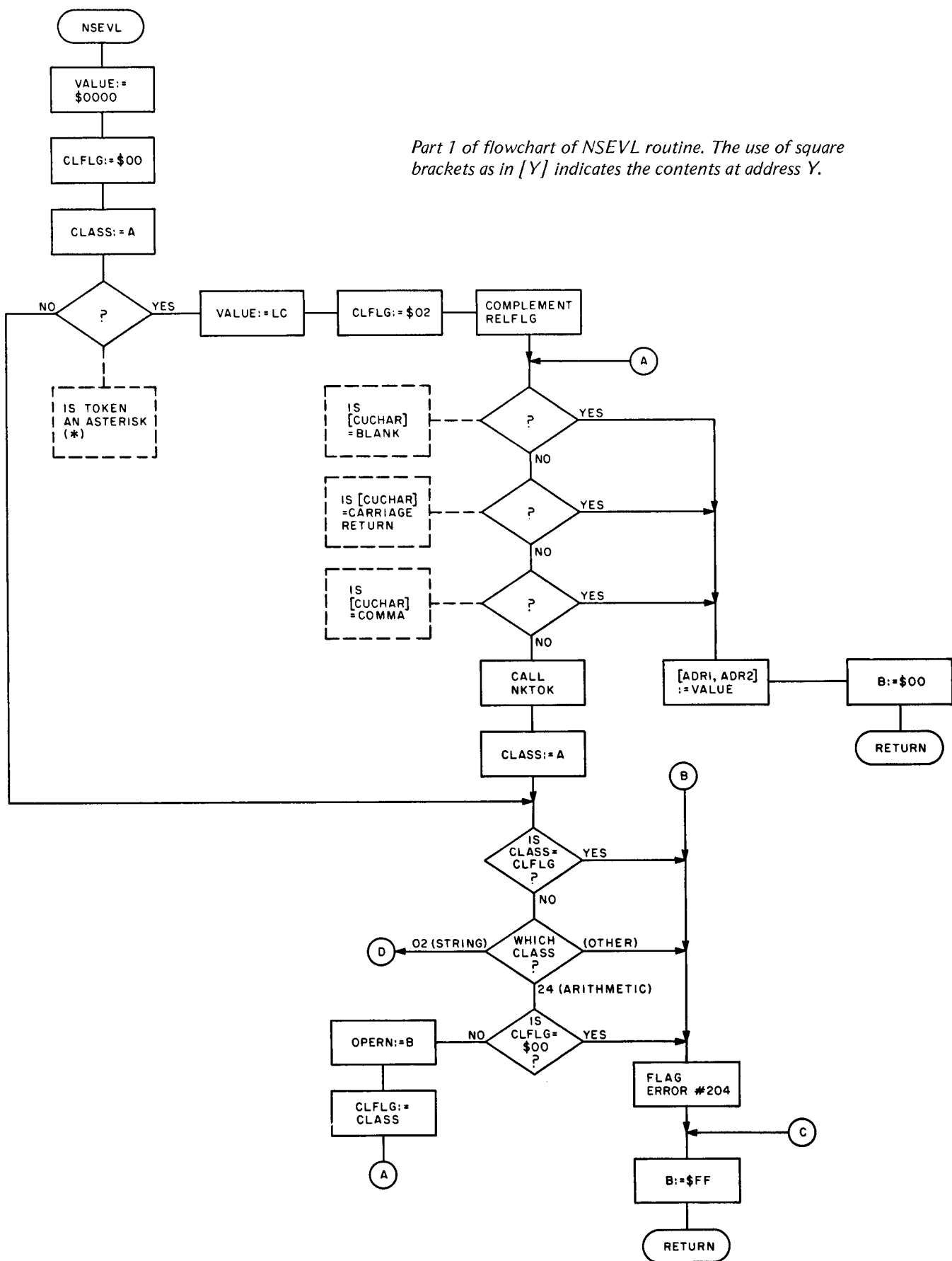
The Common flag is set if during the evaluation a symbol is found that is marked common in the Symbol Table.

On entry, the class code and the type code for the first token of the operand are in registers A and B, and the relocation flag is set to absolute (00).

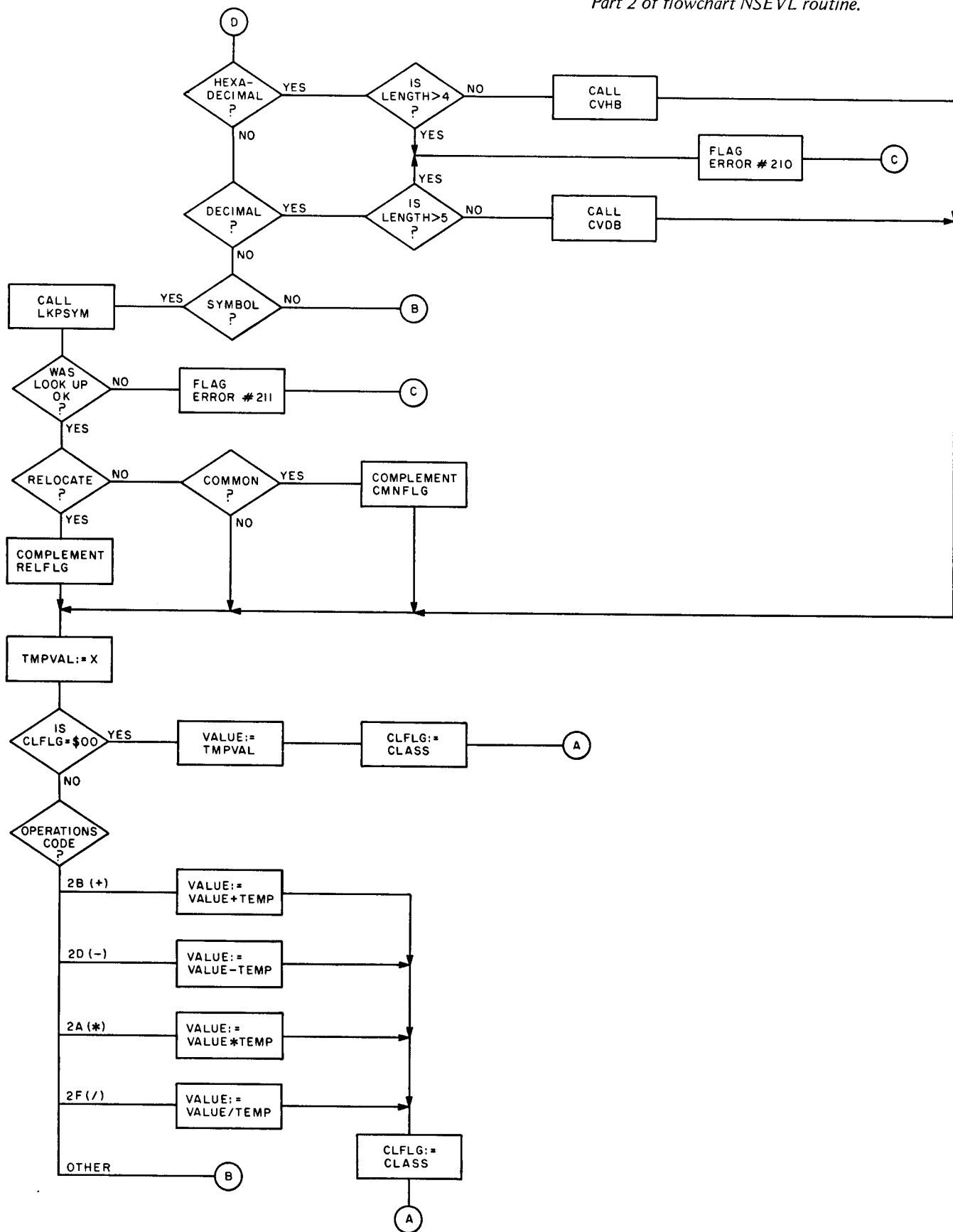
NSEVL proceeds by scanning the *operand* field and performing the indicated operations and storing the intermediate results in variables VALUE and TEMP.

On return the final sixteen bit unsigned result is in variable (ADR1, ADR2), and the B register contains a 00 if there were no errors. If there were errors the error number is in the Index register as a four digit BCD number. The relocation flag (RELFLG) is equal to 00 if the result is an absolute value, and to FF if the result is relocatable.

Calls:	ADD16, CVDB, CVHB, DIV16, LKPSYM, MPY16, NXTOK, SUB16
Called By:	POCMN, POEQU, POFCB, POFDB, POIF, PORMB, ADDR1, ADDR2, ADDR3, ADDR5, ADDR7
Flags:	CMNFLG, RELFLG
Pointers:	CLASS, CLFLG, CUCHAR, DESRC



Part 2 of flowchart NSEVL routine.



Interfacing and Using the Assembler

IO Interface Conventions

There are obviously several different methods of reading in a source program, assembling it, and finally outputting the object code. The medium used could be memory only, input from and output to cassette tapes, input from and output to floppy disk, input from tape and output to disk, etc. Included in this section on interfacing are sample IO routines for tape to tape and disk to disk systems.

Looking at the listings of the IO tape and disk routines given in Appendices J and K, notice the various entry points (such as TABLES, OUTB, WREOF, etc.) declared at the beginning. (These same names are declared as External in the main program.) These are the names of the IO routines which the user must supply for his (her) own system. Note that some of the disk routines are supplied by the authors' ICOM Floppy Disk Operating System (FDOS), while for the tape version all of the routines had to be written from scratch. Again, this may or may not be similar to the user's situation depending on the user's system configuration and software. The routines supplied in the cassette tape example could serve as a basis for any routines needed by the user.

Finally, the user should be aware that the actual lengths of this assembler and all additional tables and routines as given throughout this book assume the use of the cassette tape IO routines given in Appendix J. This means that if the user supplies his (her) own routines, the lengths and capacities described elsewhere in this book may be affected.

Tape Driver Routines

The following routines are part of a sample tape driver package. They handle the IO functions for a dual cassette tape system.

T1INZ

This routine is used to initialize and start cassette Tape1 for an input operation.

Calls: TDELY
Called By: RDBUF

T1GET

This routine is used to read a character from the input

tape, Tape1. The character is returned in register A. It checks for read errors and returns the error code in register B. If register B contains a 00 then there were no errors.

Calls: none
Called By: RDBUF

T1ISTP

This routine is used to stop Tape1 after an input operation.

Calls: none
Called By: RDBUF

T2OTZ

This routine is used to initialize and start cassette Tape2 for an output operation.

Calls: TDELY
Called By: WRITBF

T2OUT

This routine is used to output a character to Tape2. The character to be written is in register A.

Calls: none
Called By: WRITBF, T2OSTP

T2OSTP

This routine is used to stop Tape2 after a write operation.

Calls: T2OUT
Called By: WRITBF

WRITBF (Tape)

This routine writes out blocks of object code to Tape2 from the output buffer. The variable OTPTR contains the address of the last byte to be written out when the routine is called and contains the address of the first byte in the output buffer when the routine returns.

Calls: T2OTZ, T2OSTP, T2OUT
Called By: OUTB, WREOF

Disk Driver Routines

The disk drivers are all in the bootstrap Erasable Read Only Memory included in the ICOM Floppy Disk Operating System (FDOS).

- RIX — Read a byte from the disk. Byte in A register.
- WRT — Write a byte to the disk. Byte in A register. Carry flag set if End-Of-File.
- UPDATE — Close an output file.
- FDOS — Load FDOS system and pass control to it.

Assembler Loading and Execution

These instructions are written assuming two different ways to load and execute the Assembler, depending on whether the object code for the Assembler and the target program are on cassette tape or diskette. The main difference is the necessity of the ICOM Floppy Disk Operating System (FDOS) for the diskette. The procedures would be similar for any tape or disk system other than the two mentioned.

Cassette Tape Files

To load the Assembler from the cassette tape is easily accomplished if the object code for the Assembler is in absolute MIKBUG object code format. Using the MIKBUG "L" function loads the Assembler from tape. If the Assembler object code is in a relocatable format, then the Linking Loader must be utilized. For a discussion of how to do this, consult the PAPERBYTE™ book, *LINK68—Linking Loader for Motorola 6800*.

The Assembler executes as a two pass assembler, reading the input source from the cassette tape twice and, optionally, placing the generated object code onto a second cassette tape. The input source tape would go in the first cassette recorder; the object code tape, in the second tape machine.

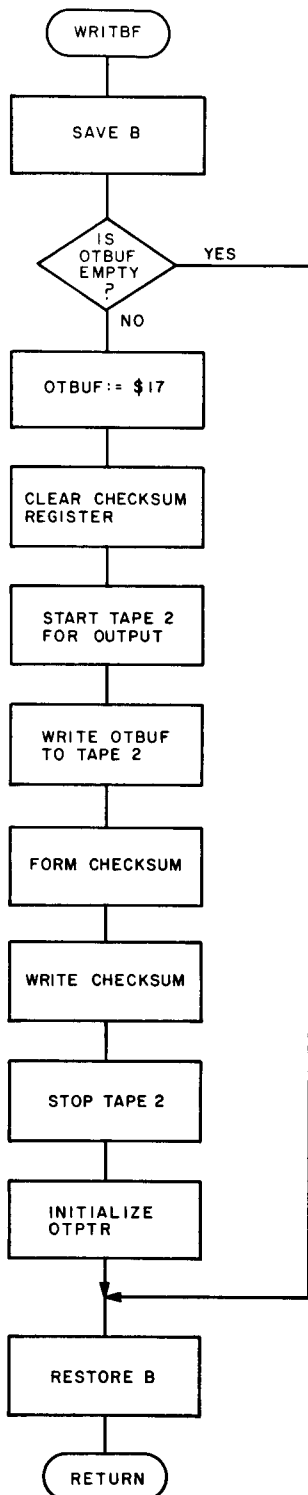
Use the MIKBUG "M" function to set the entry point of the Assembler into locations A048 and A049 (hexadecimal). If the Assembler was loaded in absolute object code form, the entry point is hexadecimal 0100. (If the Linking Loader was used to load the Assembler, then the entry point is probably different. Again, consult PAPERBYTE™ book *LINK68—Linking Loader for Motorola 6800*. If the Assembler has been relocated, care should be taken so that enough room to contain the 16 K required by the Assembler is allowed for.) Note that using the "M" function merely sets up a jump address for the start of the Assembler. If MIKBUG is not being used as a monitor, this may be accomplished in other ways.

After this setup, using the MIKBUG "G" function begins execution of the Assembler, which starts by requesting a list of the options the user desires:

ENTER OPTIONS

The options possible are:

- L — Provides a printed listing as shown in listing 1, page 15.



Flowchart of WRITBF routine.

- S — Prints a sorted Symbol Table, as shown in listing 2, page 15;
- M — All Macro expansions are printed, but only if the “L” option has also been chosen;
- O — Object code is generated.

The options desired are entered, separated by commas, and the list is terminated with a carriage return.

Example: L, O

requests that the Assembler provide a printed listing and that object code is generated, but that no Symbol Table or Macro expansions be printed.

At this point the Assembler begins Pass 1, reading the source tape in cassette 1. When the Assembler encounters an END pseudo operation in the source code, it issues the message:

REWIND TAPE & TYPE CR

At this point the user rewinds the cassette tape which contains the source and resets the controls for another read operation. Pass 1 is complete.

Pass 2 of the Assembler produces the listings, writes the object code onto cassette 2, etc. When assembly is complete, control is returned to the system monitor.

If the Assembler encounters any tape errors in the input tape it issues the warning message:

READ ERROR

and stops the tape. The user should then reposition the tape at the beginning of the block that produced the error and type a carriage return. The Assembler then will attempt to reread the block.

If an End-Of-File mark is encountered by the Assembler it types the message:

EOF: REPOSITION TAPE AND TYPE CR .

Position the tape to the beginning of the next file and type a carriage return. Consult the section entitled **Source Tape Format** for an explanation of the use of multiple files.

Diskette Files

The Assembler is located on a diskette under the name “ASMM” and is loaded and executed using the ICOM Floppy Disk Operating System (FDOS) command “RUNGO”.

Example:

RUNGO, ASMM, TEST1, TEST2

Here the input source file is TEST1 and the output object file is TEST2. Since an object file is optional, TEST2 could have been eliminated.

The Assembler requests a list of options with the statement:

ENTER OPTIONS:

The possible options are:

- L — Provides a printed listing as shown in listing 1, page 15;
- S — Prints a sorted Symbol Table as shown in listing 2, page 15;
- M — All Macro expansions are printed, but only if the “L” option has also been chosen;
- O — Object code is generated.

The options desired are entered, separated by commas, and the list is terminated with a carriage return.

Example: L,S,M

requests that a listing of the program, sorted Symbol Table, and all Macro expansions be printed, but no object code generated.

The Assembler then executes Pass 1 and Pass 2. Upon completion of the second pass, control is transferred back to the Floppy Disk Operating System.

Loading the Object Code

Loading relocatable object code generated by the Assembler is covered in detail in the companion PAPERBYTETM publication *LINK68—Linking Loader for Motorola 6800*.

Source Tape Format

The input to the Assembler is on audio tape cassette(s) in variable length blocks. The maximum length is set by the size of the input buffer in the Assembler (512 bytes).

Each line of source code is written followed by an End-of-Statement mark (a carriage return). Immediately following the last line in a block is an End-Of-Block mark (EOB, 17 hexadecimal). This is followed by a checksum character. The checksum is calculated by taking the one's complement of the summation of all the preceding bytes including the EOB. Note that lines do not span blocks.

Following the last block on the tape there is an End-Of-File (EOF) block. This block contains only one character, the EOF character (04 hexadecimal).

Thus, a file is composed of a variable number of variable length blocks followed by an EOF block.

This provision has been made so as to allow the processing of different files on different tapes, or to allow the processing of a file that is longer than the capacity of one tape side.

The user might have a set of commonly used subroutines on one tape that is used in many different programs. So long as this subroutine tape has an EOF block at the end of it, the user may use this one tape each time a different program is assembled. That is, the code on this tape does not have to be copied onto the different program tapes.

Output Object Tape Format

The output object code (relocatable) is recorded on audio cassette tape in blocks. The maximum length is set by the size of the output buffer in the Assembler (512 bytes). The format is:

Bytes 1 thru n	Relocatable object code and information for the Linking Loader.
Byte n-1	End-Of-Block (EOB) (17 hexadecimal).
Byte n-2	Checksum character byte; it is the one's complement of the summation of bytes 1 thru n.

The last block on the tape is followed by an End-Of-File block. It contains only one byte, an EOF character (04 hexadecimal).

APPENDICES

Appendix A:

Error Messages

Number	Type
0202	Opcode or label error
0204	Syntax error
0205	Label error
0206	Redefined symbol
0207	Undefined opcode
0208	Relative branch error
0210	Byte overflow
0211	Undefined symbol
0213	EQU pseudo operation error
0216	Pseudo operation error
0220	Phasing error
0221	Symbol table overflow
0223	The pseudo operation cannot be labeled
0226	The MAC pseudo operation is unlabeled
0227	MEND pseudo operation cannot be labeled
0228	Macro table overflow
0230	Macro expansion line overflow
0251	Macro nesting error
0254	IF stack overflow/underflow (nesting error)

Appendix B:

Capacities

This appendix is a summary of the various capacities of tables and stacks used in the Assembler. Some of the values are calculated from other fixed components of the Assembler, but are nonetheless set in the code. By far the largest pieces of the Assembler's total 16 K size are the Assembler's actual code, the Macro Table, and the Symbol Table. Note that the Symbol Table length is variable (see "Tables" in *The Assembler*, depending on the lengths of the particular IO routines used by the user.

Assembler (overall)	16 K
Assembler (actual code)	6 K
Character Table (CHRTAB)	64 entries, one byte per entry
If Stack (IFSTK)	8 levels of nested ifs
Macro Stack (MACSTK)	maximum of 35 nested Macro calls if no parameters on calls, 4 levels if the maximum number of parameters (8) is used on each call
Macro Table (MACTBL)	2 K, free form
Mnemonic Table (MNTAB)	86 entries, 6 bytes per entry
Symbol Table (SYMTAB)	800 symbol entries, 9 bytes per entry

Appendix C

Notes from a User: Implementation of RA6800ML

by Walter Banks, University of Waterloo

Implementation of RA6800ML is accomplished by a bootstrap procedure which ultimately results in a macro assembler specifically tailored to a unique system. This is accomplished with the use of two absolute modules presented in Appendices D and F.

In normal use RA6800ML generates relocatable object modules which are linked together by LINK68 to form a load module of absolute code. The macro assembler itself is generated as a relocatable load module requiring linking with input and output drivers to form a usable load module. This has been overcome with the use of two absolute load modules found in Appendices D and F. The ASSEMBLER load module contains a copy of the Assembler, linked to location \$0100 without any external references satisfied. The overlay modules contain external reference code for use with a standard MIKBUG-based system. This overlay is designed to facilitate easy initial implementation of RA6800ML and serve as a template for user developed software.

The macro assembler calls external routines through the use of a jump table which starts at location \$034A. Subroutine calls within the macro assembler go through the jump table to the overlayed routines and control is returned to the macro assembler with an RTS.

The IO structure of RA6800ML assumes four separate data paths. INCH and OUTCH are input and output byte routines to the user console device. GETB and OUTB are communication paths from the macro assembler to mass storage devices such as disk, tape, or paper tape. They are used to load the source code for assembling and output relocatable code modules.

The jump table calls GETB which is a subroutine used to get data from a source code input stream. The overlay prompts users to load new tapes when end-of-tape is sensed.

The calls to OUTB are used to write out the relocatable object code to the output stream. In the simple implementation these are handled by the console output routine in MIKBUG.

The calls to MONTOR and UPDATE are used to return control to the user supervisor program. UPDATE expects the user routine to close all open files. MONTOR is a direct entry to the user supervisor.

INITIO calls a routine which initializes IO devices and drivers. It is not needed in the simple overlay; however, room is left for a subroutine jump to a new program.

WREOF writes an end-of-file (\$04) to the output data stream.

A call to RESTR causes the input file to be reset at the start-of-file point. In the simple version presented here a message to rewind the tape is output and operator intervention is required.

An exception to the use of the jump table is the reference to TABLES. TABLES is used as a pointer to a data area of memory and is used only as a pointer. It must be noted that the first two locations in memory pointed to by TABLES must contain the address of TABLES+3.

Users can load a simple version of the Assembler by loading the Assembler absolute code module found in barcode form in Appendix E. The overlay package may be loaded on top of the Assembler and the combined code can be dumped to a convenient mass storage device such as a floppy disk or cassette tape. Future modifications can be made in two ways. First, the overlay package can be tailored to the unique requirements of a particular system. The absolute code may be dumped generating a new load module. Second, the whole package of Assembler and overlay can be linked from object files and a new load module generated.

APPENDIX D

RA6800ML Assembly Language Object Code in Absolute Hexadecimal Format

The listing below gives the absolute object code for the relocatable macro assembler RA6800ML in hexadecimal format. This listing can be used to manually enter the program or to verify entry of the program via the PAPERBYTE™ bar code representation given in Appendix E. Note that each line does not correspond directly to the variable length records of the bar codes, but uses a fixed length of 16 data bytes per line. The data is preceded by a 2 byte address field. Note that this program begins at hexadecimal 0100. Information on how to use this version of the Assembler to bootstrap RA6800ML for the first time is given in Appendix C, with Appendix F giving details of IO routines appropriate for the bootstrap process.

```

0100 8E A0 42 7E 04 8E 41 42 41 11 6A 1B 41 44 43 0F
0110 00 09 41 44 44 0F 00 0B 41 4E 44 0F 00 04 41 53
0120 4C 0F F9 08 41 53 52 0F F9 07 42 43 43 11 19 24
0130 42 43 53 11 19 25 42 45 51 11 19 27 42 47 45 11
0140 19 2C 42 47 54 11 19 2E 42 48 49 11 19 22 42 49
0150 54 0F 00 05 42 4C 45 11 19 2F 42 4C 53 11 19 23
0160 42 4C 54 11 19 2D 42 4D 49 11 19 2B 42 4E 45 11
0170 19 26 42 50 4C 11 19 2A 42 52 41 11 19 20 42 53
0180 52 11 19 8D 42 56 43 11 19 28 42 56 53 11 19 29
0190 43 42 41 11 6A 11 43 4C 43 11 6A 0C 43 4C 49 11
01A0 6A 0E 43 4C 52 0F F9 0F 43 4C 56 11 6A 0A 43 4D
01B0 4E 12 D4 FF 43 4D 50 0F 00 01 43 4F 4D 0F F9 03
01C0 43 50 58 10 68 8C 44 41 41 11 6A 19 44 45 43 0F
01D0 F9 0A 44 45 53 11 6A 34 44 45 58 11 6A 09 45 4E
01E0 44 13 4E FF 45 4E 54 14 D8 FF 45 4F 52 0F 00 08
01F0 45 51 55 15 51 FF 45 58 54 15 AC FF 46 43 42 16
0200 0E FF 46 43 43 16 43 FF 46 44 42 16 9A FF 49 46
0210 20 16 EE FF 49 4E 43 0F F9 0C 49 4E 53 11 6A 31
0220 49 4E 58 11 6A 08 4A 4D 50 10 E6 6E 4A 53 52 10
0230 E6 AD 4C 44 41 0F 00 06 4C 44 53 10 68 8E 4C 44
0240 58 10 68 CE 4C 53 52 0F F9 04 4D 41 43 17 2E FF
0250 4E 41 4D 18 23 FF 4E 45 47 0F F9 00 4E 49 46 18
0260 58 FF 4E 4F 50 11 6A 02 4F 52 41 0F 00 0A 50 41
0270 47 18 9D FF 50 53 48 10 48 36 50 55 4C 10 48 32
0280 52 4D 42 18 BE FF 52 4F 4C 0F F9 09 52 4F 52 0F
0290 F9 06 52 54 49 11 6A 3B 52 54 53 11 6A 39 53 42
02A0 41 11 6A 10 53 42 43 0F 00 02 53 45 43 11 6A 0D
02B0 53 45 49 11 6A 0F 53 45 56 11 6A 0B 53 54 41 0F
02C0 91 07 53 54 53 10 DA 8F 53 54 58 10 DA CF 53 55
02D0 42 0F 00 00 53 57 49 11 6A 3F 54 41 42 11 6A 16
02E0 54 41 50 11 6A 06 54 42 41 11 6A 17 54 50 41 11
02F0 6A 07 54 53 54 0F F9 0D 54 53 58 11 6A 30 54 58
0300 53 11 6A 35 57 41 49 11 6A 3E 00 00 00 04 04 04
0310 00 04 00 00 24 24 04 24 80 24 42 42 42 42 42 42
0320 42 42 42 42 00 00 00 00 00 00 80 83 83 82 82 82
0330 82 80 80 80 80 80 80 80 80 80 80 80 80 80 80
0340 80 80 81 80 80 00 00 00 00 00 7E FF FF 7E FF FF
0350 7E FF FF 7E FF FF 7E FF FF 7E FF FF 7E FF FF 7E
0360 FF FF 00 00 00 00 00 00 00 00 00 00 00 00 00
0370 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0380 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0390 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
03A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
03B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
03C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
03D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
03E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
03F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0400 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0410 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

```

```

0420 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0430 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0440 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0450 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0460 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0470 00 00 00 00 00 00 00 00 00 45 4E 54 45 52 20 4F 50
0480 54 49 4F 4E 53 3A 20 04 7E E1 AC 7E E1 D1 7F 03
0490 75 CE 04 78 BD 19 5E 7F 03 6E 73 03 6E BD 04 88
04A0 81 0D 27 26 81 4C 26 04 86 70 20 16 81 4F 26 04
04B0 86 B0 20 0E 81 53 26 04 86 D0 20 06 81 4D 26 DD
04C0 86 E0 B4 03 6E B7 03 6E 20 D3 BD 14 BE FE 03 4B
04D0 EE 00 FF 03 62 FF 08 E3 CE 08 00 FF 08 E5 CE 08
04E0 E3 BD 0C EC FE 08 E3 FF 03 64 CE 01 00 FF 08 E5
04F0 CE 08 E3 BD 0C EC FE 08 E3 FF 03 66 08 FF 03 68
0500 FF 08 E5 CE 3F FF FF 08 E3 CE 08 E3 BD 0C FD B6
0510 08 E3 F6 08 E4 CE 00 09 FF 08 E5 CE 08 E5 BD 0C
0520 A1 B7 03 6A F7 03 6B CE 00 09 FF 08 E3 CE 08 E3
0530 BD 0C 7D B7 08 E3 F7 08 E4 FE 03 68 FF 08 E5 CE
0540 08 E3 BD 0C EC FE 08 E3 FF 03 6C 86 20 FE 03 68
0550 A7 00 08 BC 03 6C 26 F8 CE 00 00 FF 03 71 FF 03
0560 88 CE 00 00 FF 04 13 BD 11 89 7F 03 87 FE 03 62
0570 FF 04 0D 7F 04 0C FE 03 66 FF 03 8A 86 FF B7 04
0580 77 CE 04 75 FF 04 75 CE 00 00 FF 03 73 FF 03 6F
0590 BD 06 68 7F 03 76 FE 03 6F 08 FF 03 6F FE 03 80
05A0 A6 00 81 2A 26 08 BD 11 89 BD 0D 0E 20 E2 7D 04
05B0 77 26 22 81 20 27 03 BD 07 F6 BD 07 F6 B6 03 7D
05C0 81 03 22 E2 BD 0A 40 8C 16 EE 27 07 8C 18 58 27
05D0 02 20 D3 6E 00 81 20 27 1D BD 07 F6 C1 01 27 0B
05E0 CE 02 05 BD 0E BB BD 0D 0E 20 A5 7C 03 76 7D 03
05F0 75 26 03 BD 08 F8 BD 07 F6 C1 01 27 0B CE 02 02
0600 BD 0E BB BD 0D 0E 20 88 BD 0A 40 81 00 27 2D BD
0610 09 57 C1 FF 27 28 C5 20 27 24 7D 04 0C 27 08 BD
0620 07 37 7D 04 0C 27 20 FF 04 0D BD 0D 0E 7C 04 0C
0630 BD 07 F6 C1 0D 26 13 F7 03 DA 20 0B 6E 00 CE 02
0640 07 BD 0E BB BD 0D 0E 7E 05 90 CE 03 DA FF 04 0F
0650 FE 03 7B A6 00 08 FF 03 7B FE 04 0F A7 00 08 FF
0660 04 0F 81 0D 26 EA 20 DF 7D 04 0C 27 09 BD 06 A5
0670 7D 04 0C 27 01 39 CE 04 15 FF 03 7E FF 03 80 BD
0680 03 53 24 06 8E A0 42 7E 13 54 81 0A 27 F1 81 00
0690 27 ED 8C 04 64 27 05 A7 00 08 20 04 C6 0D E7 00
06A0 81 0D 26 DB 39 FE 04 0D A6 00 81 17 26 0B 7A 04
06B0 0C 27 05 BD 07 9B 20 ED 39 CE 03 92 FF 03 80 FF
06C0 03 7E FF 04 11 FE 04 0D A6 00 08 FF 04 0D 81 26
06D0 27 13 FE 04 11 A7 00 08 FF 04 11 8C 03 D9 27 4B
06E0 81 0D 26 E1 39 E6 00 C0 2F 08 FF 04 0D CE 03 DA
06F0 FF 04 0F A6 00 08 81 2C 27 04 81 0D 26 F5 5A 26
0700 EF FE 04 0F A6 00 08 FF 04 0F FE 04 11 A7 00 08
0710 FF 04 11 8C 03 D9 27 13 FE 04 0F A6 00 08 FF 04
0720 0F 81 2C 27 A0 81 0D 26 E1 20 9A 86 0D A7 00 CE
0730 02 30 BD 0E BB 20 8E FF 03 8E BF 03 8C BE 03 8A
0740 CE 04 12 C6 06 A6 00 09 FF 03 90 30 09 BC 03 64
0750 27 33 FE 03 90 36 5A 26 EC CE 03 DA A6 00 81 0D
0760 27 03 08 20 F7 A6 00 FF 03 90 30 09 BC 03 64 27
0770 14 FE 03 90 36 09 8C 03 D9 26 EA BF 03 8A BE 03
0780 8C FE 03 8E 39 BE 03 66 BF 03 8A BE 03 8C CE 02
0790 51 BD 0E BB FE 03 8E 7F 04 0C 39 FF 03 8E BF 03
07A0 8C BE 03 8A CE 03 DA 32 A7 00 08 81 0D 26 F8 CE
07B0 04 0D C6 06 32 A7 00 08 5A 26 F9 BF 03 8A BE 03
07C0 8C FE 03 8E 39 36 37 E6 04 FF 07 F4 FE 07 F4 EE
07D0 00 A6 00 FE 07 F4 6C 01 26 02 6C 00 FE 07 F4 EE
07E0 02 A1 00 26 0C FE 07 F4 6C 03 26 02 6C 02 5A 26

```



```

07F0 DB 33 32 39 00 00 7F 03 7D 7C 03 7D FE 03 7E FF
0800 03 7B A6 00 08 FF 03 7E 81 20 27 F0 22 06 81 0D
0810 26 47 16 39 81 5F 23 02 20 3F BD 08 C3 85 01 27
0820 13 FE 03 7E E6 00 C1 20 27 04 C1 0D 26 0A FE 03
0830 7B E6 00 39 85 80 27 04 BD 08 73 39 85 40 27 04
0840 BD 08 5C 39 85 20 26 E6 85 04 27 0D FE 03 7B E6
0850 00 C1 24 26 D9 BD 08 98 39 4F 5F 39 FE 03 7E A6
0860 00 7C 03 7D 08 FF 03 7E BD 08 C3 85 40 26 ED C6
0870 09 20 43 FE 03 7E A6 00 7C 03 7D 08 FF 03 7E BD
0880 08 C3 85 80 26 ED 85 40 26 E9 C6 07 F1 03 7D 24
0890 03 F7 03 7D C6 01 20 1E 7F 03 7D FE 03 7E FF 03
08A0 7B FE 03 7E A6 00 7C 03 7D 08 FF 03 7E BD 08 C3
08B0 85 02 26 ED C6 03 7A 03 7D FE 03 7E 09 FF 03 7E
08C0 86 02 39 81 20 25 16 81 5F 22 12 7F 08 DF B7 08
08D0 E0 CE 08 DF BD 0C EC FE 08 DF A6 00 39 4F 39 00
08E0 00 02 EA 00 00 00 00 00 00 00 00 00 00 00 00
08F0 00 00 00 00 00 00 00 00 00 BD 09 B9 FF 03 82 A6 00
0900 81 20 26 2F FF 08 F6 CE 08 EA FF 08 F4 C6 06 FE
0910 08 F4 A6 00 08 FF 08 F4 FE 08 F6 A7 00 08 FF 08
0920 F6 5A 26 EB B6 03 73 A7 00 B6 03 74 A7 01 86 40
0930 A7 02 39 BD 09 7E 26 10 FE 03 82 86 80 AA 08 A7
0940 08 CE 02 06 BD 0E BB 39 BD 09 93 BC 08 F0 27 02
0950 20 AC CE 02 21 20 ED BD 09 B9 FF 03 82 A6 00 81
0960 20 26 03 C6 FF 39 BD 09 7E 26 08 FE 03 82 E6 08
0970 EE 06 39 BD 09 93 BC 08 F0 26 E2 C6 FF 39 FF 08
0980 E3 86 06 B7 08 E7 CE 08 EA FF 08 E5 CE 08 E3 BD
0990 07 C5 39 FE 03 82 08 08 08 08 08 08 08 08 08 BC
09A0 03 6C 26 03 FE 03 68 FF 03 82 39 FE 03 82 86 20
09B0 C6 09 A7 00 08 5A 26 FA 39 CE 20 20 FF 08 EA FF
09C0 08 EC FF 08 EE CE 08 EA FF 08 F6 FE 03 7B FF 08
09D0 F4 F6 03 7D FE 08 F4 A6 00 08 FF 08 F4 FE 08 F6
09E0 A7 00 08 FF 08 F6 5A 26 EB FE 08 EA FF 08 F0 FE
09F0 08 EC FF 08 F2 CE 08 F0 BD 0C EC FE 08 EE FF 08
0A00 F2 CE 08 F0 BD 0C EC B6 08 F0 F6 08 F1 FE 03 6A
0A10 FF 08 F2 CE 08 F2 BD 0C A1 FF 08 F0 4F C6 09 CE
0A20 08 F0 BD 0C 7D B7 08 F0 F7 08 F1 FE 03 68 FF 08
0A30 F2 CE 08 F0 BD 0C EC FE 08 F0 39 00 00 00 00 06
0A40 B6 03 7D B7 08 E7 86 57 B7 0A 3C 4F B7 0A 3B B6
0A50 0A 3B 4C B1 0A 3C 26 03 86 FF 39 F6 0A 3B FB 0A
0A60 3C 56 F7 0A 3D 4F CE 0A 3E 5A BD 0C 7D B7 08 E3
0A70 F7 08 E4 CE 01 06 FF 08 E5 CE 08 E3 BD 0C EC FE
0A80 08 E3 FF 08 E8 FE 03 7B FF 08 E5 CE 08 E3 BD 07
0A90 C5 25 0B 26 11 4F FE 08 E8 E6 05 EE 03 39 B6 0A
0AA0 3D B7 0A 3B 20 A9 B6 0A 3D B7 0A 3C 20 A1 00 00
0AB0 00 00 00 00 00 7F 0A AE 7F 0A AF 7F 0A B2 B7 0A
0AC0 B3 C1 2A 26 2D FE 03 73 FF 0A AE 86 02 B7 0A B2
0AD0 73 03 77 FE 03 7E A6 00 81 20 27 08 81 0D 27 04
0AE0 81 2C 26 08 FE 0A AE FF 0D 68 5F 39 BD 07 F6 B7
0AF0 0A B3 B1 0A B2 26 06 CE 02 04 5F 53 39 81 02 27
0B00 14 81 24 27 02 20 F0 7D 0A B2 27 EB F7 0A B4 B7
0B10 0A B2 7E 0A D3 C1 03 26 11 F6 03 7D C1 04 2F 05
0B20 CE 02 10 20 D5 BD 0B CE 20 3B C1 09 26 11 F6 03
0B30 7D C1 05 2F 05 CE 02 10 20 C0 BD 0C 2A 20 26 C1
0B40 01 27 03 7E 0A F7 BD 09 57 C5 80 26 12 C5 40 27
0B50 05 73 03 77 20 0F C5 10 27 03 73 03 78 20 06 CE
0B60 02 11 7E 0A FA FF 0A B0 7D 0A B2 26 0F FE 0A B0
0B70 FF 0A AE B6 0A B3 B7 0A B2 7E 0A D3 B6 0A B4 81
0B80 2B 26 08 CE 0A AE BD 0C EC 20 E8 81 2D 26 08 CE
0B90 0A AE BD 0C FD 20 DC 81 2A 26 15 B6 0A AE F6 0A
0BA0 AF CE 0A B0 BD 0C 7D B7 0A AE F7 0A AF 7E 0B 73
0BB0 81 2F 27 03 7E 0A F7 B6 0A AE F6 0A AF CE 0A B0

```

```

0BC0 BD 0C A1 B7 0A AE F7 0A AF 7E 0B 73 00 00 FE 03
0BD0 7B 7F 0B CC 7F 0B CD F6 03 7D 09 08 5A 26 FC F6
0BE0 03 7D BD 0C 18 B7 0B CD 5A 27 29 09 BD 0C 18 48
0BF0 48 48 48 BA 0B CD B7 0B CD 5A 27 18 09 BD 0C 18
0C00 B7 0B CC 5A 27 0E 09 BD 0C 18 48 48 48 48 BA 0B
0C10 CC B7 0B CC FE 0B CC 39 A6 00 80 30 81 09 2F 02
0C20 80 07 39 00 00 00 00 00 00 7F 0C 23 7F 0C 24
0C30 7F 0C 26 7F 0C 27 7C 0C 27 FE 03 7B 09 F6 03 7D
0C40 F7 0C 25 08 5A 26 FC FF 0C 28 E6 00 C4 0F 4F CE
0C50 0C 26 BD 0C 7D FB 0C 24 B9 0C 23 B7 0C 23 F7 0C
0C60 24 4F C6 0A CE 0C 26 BD 0C 7D B7 0C 26 F7 0C 27
0C70 FE 0C 28 09 7A 0C 25 26 CE FE 0C 23 39 37 36 A6
0C80 01 36 A6 00 36 86 10 36 30 A6 03 58 49 68 02 69
0C90 01 24 04 EB 04 A9 03 6A 00 26 F0 31 31 31 31 31
0CA0 39 37 36 A6 00 E6 01 37 36 34 30 86 01 6D 01 2B
0CB0 0B 4C 68 02 69 01 2B 04 81 11 26 F5 A7 00 A6 03
0CC0 E6 04 6F 03 6F 04 E0 02 A2 01 24 07 EB 02 A9 01
0CD0 0C 20 01 0D 69 04 69 03 64 01 66 02 6A 00 26 E6
0CE0 A7 00 E7 01 EE 00 31 31 31 32 33 39 36 37 A6 01
0CF0 E6 00 AB 03 E9 02 A7 01 E7 00 33 32 39 36 37 A6
0D00 01 E6 00 A0 03 E2 02 A7 01 E7 00 33 32 39 B6 03
0D10 6E 85 80 26 14 7D 03 75 27 0F 7D 04 0C 27 04 85
0D20 10 26 06 BD 0D 2A BD 0D 71 39 37 F6 03 87 C1 00
0D30 26 03 BD 0D 44 7C 03 87 F6 03 87 C1 3C 26 03 7F
0D40 03 87 33 39 CE 0D 4B BD 19 5E 39 0D 0A 0D 0A 0D
0D50 0A 2E 2E 2E 2E 2E 2E 2E 2E 2E 2E 2E 2E 0D 0A
0D60 0D 0A 0D 0A 04 00 00 00 00 00 00 00 00 00 20
0D70 04 CE 0D 6A B6 03 6F F6 03 70 BD 0E 5C CE 0D 6B
0D80 BD 19 5E 7D 04 0C 27 05 86 2B BD 19 85 CE 0E 59
0D90 BD 19 5E 7D 0D 65 26 0D 7D 0D 66 26 08 CE 0E 56
0DA0 BD 19 5E 20 2B CE 03 73 BD 19 6E F6 0D 66 27 20
0DB0 C1 01 27 0E CE 0D 68 BD 19 6E CE 0E 58 BD 19 5E
0DC0 20 45 CE 0D 69 BD 19 70 CE 0E 56 BD 19 5E 20 37
0DD0 F6 0D 65 26 08 CE 0E 53 BD 19 5E 20 2A CE 0D 67
0DE0 BD 19 70 C1 01 26 08 CE 0E 56 BD 19 5E 20 18 C1
0DF0 02 26 0E CE 0D 69 BD 19 70 CE 0E 59 BD 19 5E 20
0E00 06 CE 0D 68 BD 19 6E 7D 03 78 27 04 86 43 20 1D
0E10 7D 03 79 27 04 86 58 20 14 7D 03 7A 27 04 86 4E
0E20 20 0B 7D 03 77 27 04 86 52 20 02 86 20 BD 19 85
0E30 86 20 BD 19 85 FE 03 80 A6 00 36 BD 19 85 08 32
0E40 81 0D 26 F4 86 0A BD 19 85 7F 0D 66 7F 0D 65 7F
0E50 03 77 39 20 20 20 20 20 20 20 04 FF 0E 9D CE
0E60 0E 92 7F 0E 9C E0 01 A2 00 25 05 7C 0E 9C 20 F5
0E70 EB 01 A9 00 36 FF 0E 9F FE 0E 9D B6 0E 9C 8B 30
0E80 A7 00 32 08 FF 0E 9D FE 0E 9F 08 08 8C 0E 9C 26
0E90 D1 39 27 10 03 E8 00 64 00 0A 00 01 00 00 00 00
0EA0 00 00 00 2A 2A 2A 2A 20 45 52 52 4F 52 23 20 00
0EB0 00 00 20 00 00 00 00 00 20 3A 04 36 37 FF 0E A1
0EC0 B6 0E A1 8B 30 B7 0E AF B6 0E A2 44 44 44 44 8B
0ED0 30 B7 0E B0 B6 0E A2 84 0F 8B 30 B7 0E B1 CE 0E
0EE0 B3 B6 03 6F F6 03 70 BD 0E 5C CE 0E A3 BD 19 5E
0EF0 BD 0E 35 33 32 FE 03 88 08 FF 03 88 FE 0E A1 39
0F00 BD 11 89 BD 07 F6 C1 0D 26 08 CE 02 04 BD 0E BB
0F10 20 5B BD 11 B7 F6 11 84 27 F0 BD 07 F6 C1 23 26
0F20 14 73 11 85 BD 07 F6 C1 27 26 0A FE 03 7E A6 00
0F30 B7 0D 69 20 0B BD 0A B5 BD 11 D7 F6 11 85 27 0C
0F40 C6 80 F7 11 87 C6 C0 F7 11 88 20 3C BD 07 F6 BD
0F50 11 C4 26 2A 7D 03 78 26 0A 7D 03 77 26 05 F6 0D
0F60 68 27 0F C6 B0 F7 11 87 C6 F0 F7 11 88 BD 12 5E
0F70 20 19 C6 90 F7 11 87 C6 D0 F7 11 88 20 0A C6 A0
0F80 F7 11 87 C6 E0 F7 11 88 BD 12 13 BD 12 C2 7E 05

```

```

0F90 90 BD 11 89 BD 07 F6 C1 0D 26 08 CE 02 04 BD 0E
0FA0 BB 20 32 BD 11 B7 F6 11 84 27 F0 BD 07 F6 BD 0A
0FB0 B5 BD 11 D7 BD 07 F6 BD 11 C4 26 2A 7D 03 78 26
0FC0 0A 7D 03 77 26 05 F6 0D 68 27 0F C6 B0 F7 11 87
0FD0 C6 F0 F7 11 88 BD 12 5E 20 19 C6 90 F7 11 87 C6
0FE0 D0 F7 11 88 20 0A C6 A0 F7 11 87 C6 E0 F7 11 88
0FF0 BD 12 13 BD 12 C2 7E 05 90 BD 11 89 BD 07 F6 C1
1000 0D 26 08 CE 02 04 BD 0E BB 20 2A BD 11 B7 7D 11
1010 84 27 0F C6 40 F7 11 87 C6 50 F7 11 88 BD 11 EA
1020 20 20 BD 0A B5 BD 11 D7 BD 07 F6 BD 11 C4 26 0A
1030 C6 70 F7 11 87 BD 12 7C 20 08 C6 60 F7 11 87 BD
1040 12 31 BD 12 C2 7E 05 90 BD 11 89 BD 07 F6 BD 11
1050 B7 7D 11 84 26 06 CE 02 04 BD 0E BB 7C 11 88 BD
1060 11 EA BD 12 C2 7E 05 90 BD 11 89 BD 07 F6 C1 0D
1070 26 08 CE 02 40 BD 0E BB 20 46 C1 23 26 19 73 11
1080 85 BD 07 F6 C1 27 26 0F FE 03 7E A6 00 B7 0D 68
1090 A6 01 B7 0D 69 20 29 BD 0A B5 BD 11 D7 F6 11 85
10A0 27 02 20 1C BD 07 F6 BD 11 C4 26 20 7D 03 78 26
10B0 0A 7D 03 77 26 05 F6 0D 68 27 0A C6 30 F7 11 87
10C0 BD 12 7C 20 0F C6 10 F7 11 87 20 05 C6 20 F7 11
10D0 87 BD 12 31 BD 12 C2 7E 05 90 BD 11 89 BD 07 F6
10E0 C1 0D 26 B3 20 8C BD 11 89 BD 07 F6 C1 0D 26 08
10F0 CE 02 04 BD 0E BB 20 0E BD 0A B5 BD 11 D7 BD 07
1100 F6 BD 11 C4 26 0A C6 10 F7 11 87 BD 12 7C 20 03
1110 BD 12 31 BD 12 C2 7E 05 90 BD 11 89 BD 07 F6 C1
1120 0D 26 08 CE 02 04 BD 0E BB 20 36 7D 03 75 27 31
1130 BD 0A B5 BD 11 D7 FE 03 73 08 08 FF 03 85 B6 0D
1140 69 F6 0D 68 B0 03 86 F2 03 85 C1 FF 26 03 4D 2B
1150 0D C1 00 26 03 4D 2A 06 CE 02 08 BD 0E BB B7 0D
1160 69 BD 12 31 BD 12 C2 7E 05 90 BD 11 89 7D 03 75
1170 27 03 BD 19 2B 7C 0D 65 7C 03 84 BD 0D 0E BD 12
1180 C2 7E 05 90 00 00 00 00 00 7F 03 84 7F 03 77 7F
1190 03 78 7F 03 79 7F 03 7A 7F 0D 66 F7 0D 67 7F 0D
11A0 65 7F 11 84 7F 11 85 7F 11 86 7F 0D 68 7F 0D 69
11B0 7F 11 87 7F 11 88 39 C1 41 27 05 C1 42 27 01 39
11C0 F7 11 84 39 C1 2C 26 0B BD 07 F6 C1 58 26 04 73
11D0 11 86 39 7F 11 86 39 C1 FF 26 0E 7D 03 75 27 03
11E0 BD 0E BB 7F 0D 68 73 0D 68 39 7D 07 52 72 0F 62
11F0 0D 67 B6 11 84 27 0F 81 42 27 05 FA 11 87 20 03
1200 FA 11 88 F7 0D 67 BD 19 2B 7C 0D 65 BD 0D 0E 7C
1210 03 84 39 7D 03 75 27 3F F6 0D 67 B6 11 84 27 25
1220 81 42 27 05 FA 11 87 20 03 FA 11 88 F7 0D 67 20
1230 14 7D 03 75 27 21 7F 03 77 7F 03 78 F6 0D 67 FA
1240 11 87 F7 0D 67 BD 19 2B F6 0D 69 BD 19 2B 7C 0D
1250 65 7C 0D 65 BD 0D 0E 7C 03 84 7C 03 84 39 7D 03
1260 75 27 55 F6 0D 67 B6 11 84 27 1F 81 42 27 05 FA
1270 11 87 20 03 FA 11 88 F7 0D 67 20 0E 7D 03 75 27
1280 37 F6 0D 67 FA 11 87 F7 0D 67 BD 19 2B F6 0D 68
1290 BD 19 2B F6 0D 69 BD 19 2B 7D 03 78 27 04 C6 4D
12A0 20 07 7D 03 77 27 05 C6 52 BD 19 3F 7C 0D 65 7C
12B0 0D 65 7C 0D 65 BD 0D 0E 7C 03 84 7C 03 84 7C 03
12C0 84 39 B6 03 74 F6 03 73 BB 03 84 C9 00 B7 03 74
12D0 F7 03 73 39 BD 11 89 BD 19 17 BD 07 F6 C1 01 27
12E0 08 CE 02 16 BD 0E BB 20 5D 7D 03 75 26 41 BD 08
12F0 F8 FE 03 82 FF 13 4C BD 07 F6 C1 2C 26 E3 BD 07
1300 F6 BD 0A B5 C1 FF 27 DC FE 13 4C 86 BF A4 08 8A
1310 10 A7 08 B6 04 13 A7 06 B6 04 14 A7 07 B6 0D 69
1320 F6 0D 68 BB 04 14 F9 04 13 B7 04 14 F7 04 13 BD
1330 09 57 FE 03 82 EE 06 FF 0D 68 73 0D 66 73 03 78
1340 7C 0D 65 7C 0D 65 BD 0D 0E 7E 05 90 00 00 BD 11
1350 89 BD 19 17 7D 03 75 26 0F FE 03 73 FF 03 71 73

```

```

1360 03 75 BD 03 5F 7E 05 67 FE 03 71 BC 03 73 27 06
1370 CE 02 20 BD 0E BB B6 03 6E 85 80 26 06 BD 0D 0E
1380 BD 14 BE B6 03 6E 85 20 27 03 7E 14 4D CE 14 C8
1390 FF 14 D1 7F 14 D5 FE 03 68 20 09 08 08 08 08
13A0 08 08 08 08 BC 03 6C 26 0B 7D 14 D5 27 03 7E 13
13B0 EB 7E 14 4D E6 00 C1 20 27 E1 E6 08 C1 FF 27 DB
13C0 FF 14 D3 FF 08 E5 FE 14 D1 FF 08 E3 C6 06 F7 08
13D0 E7 CE 08 E3 BD 07 C5 22 05 FE 14 D3 20 BD FE 14
13E0 D3 FF 14 D1 C6 FF F7 14 D5 20 B0 BD 0D 2A C6 06
13F0 FE 14 D1 A6 00 BD 19 85 08 5A 26 F7 86 20 BD 19
1400 85 BD 19 6E FF 14 D6 E6 00 C5 40 27 05 86 52 BD
1410 19 85 C5 20 27 05 86 4D BD 19 85 C5 10 27 05 86
1420 43 BD 19 85 C5 08 27 05 86 58 BD 19 85 C5 04 27
1430 05 86 4E BD 19 85 E6 00 2A 06 CE 14 8A BD 19 5E
1440 FE 14 D6 C6 FF E7 00 BD 14 BE 7E 13 8D BD 14 BE
1450 BD 14 BE CE 14 A1 B6 03 88 F6 03 89 BD 0E 5C CE
1460 14 95 BD 19 5E BD 14 BE BD 14 BE CE 14 AE BD 19
1470 5E CE 04 13 BD 19 6E BD 14 BE B6 03 6E 85 40 26
1480 06 BD 03 59 7E 03 4D 7E 03 50 20 52 45 44 45 46
1490 49 4E 45 44 04 54 48 45 52 45 20 57 45 52 45 3A
14A0 20 00 00 00 00 00 20 45 52 52 4F 52 53 04 43 4F
14B0 4D 4D 4F 4E 20 4C 45 4E 47 54 48 3D 20 04 CE 14
14C0 C5 BD 19 5E 39 0D 0A 04 5B 5B 5B 5B 5B 5B 00 00
14D0 00 00 00 00 00 00 00 00 BD 11 89 BD 19 17 BD 07
14E0 F6 C1 01 27 08 CE 02 16 BD 0E BB 20 39 7D 03 75
14F0 27 43 BD 09 57 C1 FF 26 08 CE 02 11 BD 0E BB 20
1500 25 FF 0D 68 FE 03 82 A6 08 8A 04 A7 08 BD 15 38
1510 F6 0D 68 BD 19 2B F6 0D 69 BD 19 2B C6 52 BD 19
1520 3F C6 4E BD 19 3F 73 0D 66 73 03 7A 7C 0D 65 7C
1530 0D 65 BD 0D 0E 7E 05 90 FE 03 82 86 06 E6 00 36
1540 FF 15 4F BD 19 2B 32 FE 15 4F 08 4A 26 EF 39 00
1550 00 BD 11 89 FE 03 82 FF 15 AA 7D 03 76 26 08 CE
1560 02 13 BD 0E BB 20 3D BD 07 F6 C1 0D 26 05 CE 02
1570 16 20 EF BD 0A B5 C1 FF 27 E8 7D 03 75 26 1C FE
1580 15 AA A6 08 7D 03 77 26 04 84 BF 20 02 8A 40 A7
1590 08 B6 0D 69 A7 07 B6 0D 68 A7 06 73 0D 66 7C 0D
15A0 65 7C 0D 65 BD 0D 0E 7E 05 90 00 00 BD 11 89 BD
15B0 19 17 BD 07 F6 C1 01 27 0B CE 02 16 BD 0E BB BD
15C0 0D 0E 20 47 7C 03 84 7C 03 84 7C 03 84 7D 03 75
15D0 26 0E BD 08 F8 FE 03 82 A6 08 8A 08 A7 08 20 28
15E0 C6 7E F7 0D 67 BD 19 2B BD 09 57 BD 15 38 C6 58
15F0 BD 19 3F 7F 0D 68 7F 0D 69 7C 0D 65 7C 0D 65 7C
1600 0D 65 73 03 79 BD 0D 0E BD 12 C2 7E 05 90 BD 11
1610 89 BD 07 F6 C1 0D 26 08 CE 02 16 BD 0E BB 20 1A
1620 BD 0A B5 BD 11 D7 7C 03 84 7D 03 75 27 0F F6 0D
1630 69 BD 19 2B 7C 0D 65 7C 0D 66 BD 0D 0E BD 12 C2
1640 7E 05 90 BD 11 89 BD 07 F6 C1 27 27 08 CE 02 04
1650 BD 0E BB 20 3C FE 03 7E E6 00 C1 0D 27 EF F7 0D
1660 69 7D 03 75 27 03 BD 19 2B FE 03 7E 08 FF 03 7E
1670 7C 03 84 E6 00 C1 27 27 06 C1 0D 26 E4 20 0C E6
1680 01 C1 27 26 06 08 FF 03 7E 20 D6 7C 0D 66 7C 0D
1690 65 BD 0D 0E BD 12 C2 7E 05 90 BD 11 89 BD 07 F6
16A0 C1 0D 26 08 CE 02 16 BD 0E BB 20 39 BD 0A B5 BD
16B0 11 D7 7C 03 84 7C 03 84 7D 03 75 27 2B F6 0D 68
16C0 BD 19 2B F6 0D 69 BD 19 2B 7D 03 77 27 04 C6 52
16D0 20 07 7D 03 78 27 05 C6 4D BD 19 3F 7C 0D 65 7C
16E0 0D 65 73 0D 66 BD 0D 0E BD 12 C2 7E 05 90 BD 11
16F0 89 BD 19 17 BD 07 F6 C1 0D 26 08 CE 02 16 BD 0E
1700 BB 20 23 BD 0A B5 C1 FF 27 F1 BD 18 67 7D 04 77
1710 27 14 7D 0D 68 26 0A 7D 0D 69 26 05 7F 04 77 20
1720 05 86 FF B7 04 77 BD 0D 0E 7E 05 90 00 00 BD 11

```

```

1730 89 BD 0D 0E 7F 17 2C 7F 17 2D 7D 03 75 26 30 7D
1740 03 76 26 0B 73 17 2D CE 02 26 BD 0E BB 20 20 FE
1750 03 82 86 20 A7 08 B6 04 0D A7 06 B6 04 0E A7 07
1760 BD 07 F6 FE 03 7B A6 00 81 43 26 03 73 17 2C BD
1770 06 76 FE 03 6F 08 FF 03 6F BD 0D 0E FE 03 80 A6
1780 00 81 2A 26 0A 7D 17 2C 27 E5 BD 17 E3 20 E0 7F
1790 03 76 FE 03 80 A6 00 81 20 27 06 BD 07 F6 73 03
17A0 76 BD 07 F6 86 04 B7 08 E7 FE 03 7B FF 08 E3 CE
17B0 18 1F FF 08 E5 CE 08 E3 BD 07 C5 26 0D 7D 03 76
17C0 27 0E CE 02 27 BD 0E BB 20 06 BD 17 E3 7E 17 6F
17D0 7D 03 75 26 0B 86 17 FE 04 0D A7 00 08 FF 04 0D
17E0 7E 05 90 7D 03 75 26 36 7D 17 2D 26 31 FE 03 80
17F0 FF 03 7E FE 03 7E A6 00 08 FF 03 7E FE 04 0D BC
1800 03 64 26 10 86 0D A7 00 08 FF 04 0D CE 02 28 BD
1810 0E BB 20 0A A7 00 08 FF 04 0D 81 0D 26 D5 39 4D
1820 45 4E 44 BD 11 89 BD 19 17 BD 07 F6 C1 01 27 09
1830 CE 02 16 BD 0E BB 7E 15 26 7D 03 75 26 06 BD 08
1840 F8 7E 14 ED F6 04 13 BD 19 2B F6 04 14 BD 19 2B
1850 C6 50 BD 19 3F 7E 14 ED BD 11 89 BD 19 17 BD 18
1860 7B BD 0D 0E 7E 05 90 BF 03 8C FE 04 75 8C 04 6D
1870 27 1D BE 04 75 B6 04 77 36 20 1B BF 03 8C FE 04
1880 75 8C 04 75 27 09 BE 04 75 32 B7 04 77 20 07 CE
1890 02 54 BD 0E BB 39 BF 04 75 BE 03 8C 39 BD 11 89
18A0 BD 19 17 7D 03 75 27 13 F6 03 87 27 0E C6 3C F0
18B0 03 87 BD 14 BE 5A 26 FA 7F 03 87 7E 05 90 BD 11
18C0 89 BD 07 F6 C1 0D 26 08 CE 02 16 BD 0E BB 20 18
18D0 BD 0A B5 C1 FF 27 F4 7D 03 75 27 0F BD 19 00 7C
18E0 0D 65 7C 0D 65 73 0D 66 BD 0D 0E B6 03 74 F6 03
18F0 73 BB 0D 69 F9 0D 68 B7 03 74 F7 03 73 7E 05 90
1900 5F FE 0D 68 FF 03 85 BD 19 2B FE 03 85 09 27 06
1910 FF 03 85 5F 20 F1 39 7D 03 76 27 0E 7D 03 75 26
1920 03 BD 09 AB CE 02 23 BD 0E BB 39 B6 03 6E 85 40
1930 26 0C 17 8D 16 BD 03 56 17 8D 14 BD 03 56 39 B6
1940 03 6E 85 40 26 F8 17 BD 03 56 39 44 44 44 44 84
1950 0F 8B 30 81 39 23 02 8B 07 39 BD 19 85 08 A6 00
1960 81 04 26 F6 39 A6 00 8D 0E A6 00 08 20 0D 8D F5
1970 8D F3 86 20 7E 19 85 44 44 44 44 84 0F 8B 30 81
1980 39 23 02 8B 07 36 BD 04 8B 32 81 0A 26 0E 36 37
1990 C6 08 86 00 BD 04 8B 5A 26 F8 33 32 39 DE DE DE
19A0 DE DE DE DE DE DE DE DE DE DE DE DE DE DE DE
19B0 DE

```


APPENDIX E

PAPERBYTE™ Bar Code Representation of RA6800ML in Absolute Format

Beginning on the following page is a complete machine readable representation (PAPERBYTE™ bar codes) of the object code for the Hemenway relocatable macro assembler RA6800ML. This object code was created by assembling the Assembler. See appendix G for a listing of the 6800 assembly language source code of the Assembler.

This representation uses the absolute loader format, in which each bar code frame (one line of bar codes running from top to bottom of the page) contains a 2 byte address followed by data which is loaded in ascending order starting at that address. A hexadecimal listing that can be used to verify the input from bar codes is given in Appendix D. For details on the frame format and absolute loader format used in this and other PAPERBYTE™ books, see PAPERBYTE publication *Bar Code Loader* by Ken Budnick. The book contains a brief history on bar codes, a general bar code loader algorithm with flowcharts, and complete program listings for 6800, 6502, and 8080 or Z-80 based systems.

Information on how to use this version of the Assembler to bootstrap RA6800ML for the first time is given in Appendix C, with Appendix F giving details of IO routines appropriate for the bootstrap process.

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0	1	2	4	5	6	A	B	D	E	F	I	J	K	L	M	N	P	R	S
0	6	B	O	5	A	F	G	H	C	8	9	E	T	V	W	X	Y	Z	4

[illegible]

1 2 0	1 2 1	1 2 2	1 2 3	1 2 4	1 2 5	1 2 6	1 2 7	1 2 8	1 2 9	1 3 0	1 3 1	1 3 2	1 3 3	1 3 4	1 3 5	1 3 6	1 3 7	1 3 8	1 3 9	1 4 0	1 4 1	1 4 2	1 4 3	1 4 4	1 4 5	1 4 6	1 4 7	1 4 8	1 4 9	1 5 0	1 5 1	1 5 2	1 5 3	1 5 4	1 5 5	1 5 6	1 5 7	1 5 8	1 5 9
0 A B A	0 A C D	0 A E I	0 A F 4	0 B 0 9	0 B 1 D	0 B 3 1	0 B 4 6	0 B 5 A	0 B 6 E	0 B 8 2	0 B 9 6	0 B A A	0 B B D	0 B D 1	0 B E 4	0 B F 8	0 C 0 C	0 C 2 1	0 C 3 6	0 C 4 A	0 C 5 E	0 C 7 2	0 C 8 6	0 C 9 B	0 C B 0	0 C C 5	0 C D A	0 C E E	0 D 0 3	0 D 1 8	0 D 2 C	0 D 4 0	0 D 5 4	0 D 6 9	0 D 7 E	0 D 9 2	0 D A 6	0 D B A	0 D C E

1 2 0	1 2 1	1 2 2	1 2 3	1 2 4	1 2 5	1 2 6	1 2 7	1 2 8	1 2 9	1 3 0	1 3 1	1 3 2	1 3 3	1 3 4	1 3 5	1 3 6	1 3 7	1 3 8	1 3 9	1 4 0	1 4 1	1 4 2	1 4 3	1 4 4	1 4 5	1 4 6	1 4 7	1 4 8	1 4 9	1 5 0	1 5 1	1 5 2	1 5 3	1 5 4	1 5 5	1 5 6	1 5 7	1 5 8	1 5 9
0 A B A	0 A C D	0 A E I	0 A F 4	0 B 0 9	0 B 1 D	0 B 3 1	0 B 4 6	0 B 5 A	0 B 6 E	0 B 8 2	0 B 9 6	0 B A A	0 B B D	0 B D 1	0 B E 4	0 B F 8	0 C 0 C	0 C 2 1	0 C 3 6	0 C 4 A	0 C 5 E	0 C 7 2	0 C 8 6	0 C 9 B	0 C B 0	0 C C 5	0 C D A	0 C E E	0 D 0 3	0 D 1 8	0 D 2 C	0 D 4 0	0 D 5 4	0 D 6 9	0 D 7 E	0 D 9 2	0 D A 6	0 D B A	0 D C E

1	6	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
6	6	6	6	6	6	6	6	7	7	7	7	7	7	8	8	8	8	9	9
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9

2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
8	8	8	8	8	8	8	8	8	9	9	9	9	9	9	9	9	9	9	9
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9

APPENDIX F

Input and Output Routines for RA6800ML in Absolute Format with PAPERBYTE™ Bar Code Representation

These overlay modules contain external reference code to the relocatable macro assembler RA6800ML for use with a standard MIKBUG-based system. This overlay is designed to facilitate easy initial implementation of RA6800ML and serve as a template for user developed software. These routines can be used in conjunction with the version of RA6800ML given in Appendices D and E to bootstrap RA6800ML for the first time. Details of the bootstrap process are given in Appendix C. On page 93 is the machine readable representation (PAPERBYTE™ bar codes) of the object code of the IO routines listed below. The representation uses the absolute loader format, in which each bar code frame (one line of bars running from top to bottom of the page) contains a 2 byte address followed by data which is loaded in ascending order starting at that address. For details on the frame format and absolute loader format used in this and other PAPERBYTE™ books, see the PAPERBYTE publication *Bar Code Loader* by Ken Budnick. This book contains a brief history on bar codes, a general bar code loader algorithm with flowcharts, and complete program listings for 6800, 6502, and 8080 or Z-80 based systems.

00001			NAM	ASSIO	
00003	0100	START	EQU	\$0100	START OF THE ASSEMBLER
00005	E1AC	INCH	EQU	\$E1AC	INPUT CHAR (MIKBUG)
00006	E1D1	OUTCH	EQU	\$E1D1	OUTPUT CHAR (MIKBUG)
00007	E1D1	OUTB	EQU	\$E1D1	OUT DATA CHAR FROM ASSEMBLER
00008	E0E3	MONTOR	EQU	\$E0E3	EXIT BACK TO MONITOR (MIKBUG)
00009	E0E3	UPDATE	EQU	\$E0E3	CLOSE OUTPUT FILES ,EXIT
00011	034A		ORG	START+\$24A	
00013	034A 7E 1A0E		JMP	TABLES	START OF SYMBOL TABLE
00014	034D 7E E0E3		JMP	UPDATE	CLOSE AN OUTPUT FILE
00015	0350 7E E0E3		JMP	MONTOR	MONITOR START ADDRESS
00016	0353 7E 19A4		JMP	GETB	READ A BYTE FROM RELOCATION
00017		*			INPUT STRING
00018	0356 7E E1D1		JMP	OUTB	WRITE A BYTE
00019	0359 7E 19C2		JMP	WREOF	WRITE EOF ON SAVE FILE
00020	035C 7E 19A0		JMP	INITIO	INIT IO DEVICES
00021	035F 7E 19C8		JMP	RESTR	REWIND AN INPUT FILE
00023	0488		ORG	START+\$388	
00025	0488 7E E1AC	INEEE	JMP	INCH	INPUT CHAR TO ACC A
00026	048B 7E E1D1	OUTEEE	JMP	OUTCH	OUTPUT BYTE IN ACC A
00028	1961	PDATA1	EQU	START+\$1861	PRINT CHAR STRING
00029	14BE	CRLF	EQU	START+\$13BE	PRINT <CR> <LF>
00031	19A0		ORG	START+\$18A0	START AT THE END OF
00032		*			THE ASSEMBLER
00034	19A0 01	INITIO	NOP		INITIALIZE I/O DRIVERS
00035	19A1 01		NOP		
00036	19A2 01		NOP		
00037	19A3 39		RTS		
00039	19A4 FF 1A0C	GETB	STX	DXSV	SAVE INDEX REGISTER
00040	19A7 BD 0488	GET1	JSR	INEEE	INPUT A DATA CHARACTER
00041	19AA 81 04		CMP A	#\$04	IS IT END OF FILE
00042	19AC 26 0F		BNE	XIT	NO EXIT
00043	19AE CE 19F3		LDX	#EOF	YES PRINT EOF MESSAGE ON
00044		*			CONSOLE
00045	19B1 BD 1961		JSR	PDATA1	

00046	19B4	BD 0488	RD6	JSR	INEEE	FOR CONSOLE RESPONSE
00047	19B7	81 0D		CMP A	#\$0D	<CR> START READING NEXT TAPE
00048	19B9	27 EC		BEQ	GET1	
00049	19BB	20 F7		BRA	RD6	
00050	19BD	FE 1A0C	X1T	LDX	DXSV	RESTORE INDEX REGISTER
00051	19C0	0C		CLC		CLEAR CARRY NOT EOF
00052	19C1	39		RTS		
00054	19C2	96 04	WREOF	LDA A	4	LOAD ASCII EOF
00055	19C4	BD E1D1		JSR	OUTB	OUTPUT IT TO DATA STREAM
00056	19C7	39		RTS		
00058	19C8	CE 19D9	RESTR	LDX	#REWIND	
00059	19CB	BD 1961		JSR	PDATA1	
00060	19CE	BD 0488	BACK	JSR	INEEE	
00061	19D1	81 0D		CMP A	#\$0D	IS IT A CR?
00062	19D3	26 F9		BNE	BACK	
00063	19D5	BD 14BE		JSR	CRLF	
00064	19D8	39		RTS		
00066	19D9	0D0A	REWIND	FDB	\$0D0A	
00067	19DB	524557		FCC	/REWIND TAPE TYPE CR/	
		19DE				
		19E1				
		19E4				
		19E7				
		19EA				
		19ED				
		19F0				
00068	19F2	04		FCB	4	
00069	19F3	454F46	EOF	FCC	/EOF: NEXT TAPE,TYPE CR/	
		19F6				
		19F9				
		19FC				
		19FF				
		1A02				
		1A05				
		1A08				
00070	1A09	0D0A		FDB	\$0D0A	
00071	1A0B	04		FCB	4	
00073	1A0C	0002	DXSV	RMB	2	SAVE SPACE FOR TEMP STORAGE OF
00074			*			THE INDEX REGISTER
00076	1A0E	1A10	TABLES	FDB	*+2	START OF SYMBOL TABLE
00078				END		

0	0	0	1	1	1	1	1	1	1
3	3	4	9	9	9	9	9	A	A
4	5	8	A	B	C	D	F	0	0
A	E	8	0	4	8	C	1	6	E

1
 2
 3
 4
 5
 6
 7
 8
 9
 10
 11
 12
 13
 14
 15
 16
 17
 18
 19
 20
 21
 22
 23
 24
 25
 26
 27
 28
 29
 30
 31
 32
 33
 34
 35
 36
 37
 38
 39
 40
 41
 42
 43
 44
 45
 46
 47
 48
 49
 50
 51
 52
 53
 54
 55
 56
 57
 58
 59
 60
 61
 62
 63
 64
 65
 66
 67
 68
 69
 70
 71
 72
 73
 74
 75
 76
 77
 78
 79
 80
 81
 82
 83
 84
 85
 86
 87
 88
 89
 90
 91
 92
 93
 94
 95
 96
 97
 98
 99
 100
 101
 102
 103
 104
 105
 106
 107
 108
 109
 110
 111
 112
 113
 114
 115
 116
 117
 118
 119
 120
 121
 122
 123
 124
 125
 126
 127
 128
 129
 130
 131
 132
 133
 134
 135
 136
 137
 138
 139
 140
 141
 142
 143
 144
 145
 146
 147
 148
 149
 150
 151
 152
 153
 154
 155
 156
 157
 158
 159
 160
 161
 162
 163
 164
 165
 166
 167
 168
 169
 170
 171
 172
 173
 174
 175
 176
 177
 178
 179
 180
 181
 182
 183
 184
 185
 186
 187
 188
 189
 190
 191
 192
 193
 194
 195
 196
 197
 198
 199
 200
 201
 202
 203
 204
 205
 206
 207
 208
 209
 210
 211
 212
 213
 214
 215
 216
 217
 218
 219
 220
 221
 222
 223
 224
 225
 226
 227
 228
 229
 230
 231
 232
 233
 234
 235
 236
 237
 238
 239
 240
 241
 242
 243
 244
 245
 246
 247
 248
 249
 250
 251
 252
 253
 254
 255
 256
 257
 258
 259
 260
 261
 262
 263
 264
 265
 266
 267
 268
 269
 270
 271
 272
 273
 274
 275
 276
 277
 278
 279
 280
 281
 282
 283
 284
 285
 286
 287
 288
 289
 290
 291
 292
 293
 294
 295
 296
 297
 298
 299
 300
 301
 302
 303
 304
 305
 306
 307
 308
 309
 310
 311
 312
 313
 314
 315
 316
 317
 318
 319
 320
 321
 322
 323
 324
 325
 326
 327
 328
 329
 330
 331
 332
 333
 334
 335
 336
 337
 338
 339
 340
 341
 342
 343
 344
 345
 346
 347
 348
 349
 350
 351
 352
 353
 354
 355
 356
 357
 358
 359
 360
 361
 362
 363
 364
 365
 366
 367
 368
 369
 370
 371
 372
 373
 374
 375
 376
 377
 378
 379
 380
 381
 382
 383
 384
 385
 386
 387
 388
 389
 390
 391
 392
 393
 394
 395
 396
 397
 398
 399
 400
 401
 402
 403
 404
 405
 406
 407
 408
 409
 410
 411
 412
 413
 414
 415
 416
 417
 418
 419
 420
 421
 422
 423
 424
 425
 426
 427
 428
 429
 430
 431
 432
 433
 434
 435
 436
 437
 438
 439
 440
 441
 442
 443
 444
 445
 446
 447
 448
 449
 450
 451
 452
 453
 454
 455
 456
 457
 458
 459
 460
 461
 462
 463
 464
 465
 466
 467
 468
 469
 470
 471
 472
 473
 474
 475
 476
 477
 478
 479
 480
 481
 482
 483
 484
 485
 486
 487
 488
 489
 490
 491
 492
 493
 494
 495
 496
 497
 498
 499
 500
 501
 502
 503
 504
 505
 506
 507
 508
 509
 510
 511
 512
 513
 514
 515
 516
 517
 518
 519
 520
 521
 522
 523
 524
 525

93

APPENDIX G

Assembly Language Source Listing of RA6800ML

This assembly was executed using the relocatable macro assembler RA6800ML. The object code in the assembly listing can be used without relocation if the program is loaded at location zero (hexadecimal) in memory. When creating a final object module for the Assembler, hand entered overlays for the Motorola MIKBUG monitor or the ICOM Floppy Disk Operating System IO routines will be necessary. The routines given in Appendices J and K can be used directly with their respective operating systems, or as guidelines for coding patches to interface other monitor programs.

```

0000 0000  N      NAM  ASM  RESIDENT MACRO ASSEMBLER
          *      (RELOCATING AND LINKING)
          *      VERSION 1.0
          *
          * C COPYRIGHT 1977 JACK E. HEMENWAY
          * BOSTON MASS. ALL RIGHTS RESERVED
          *
0000 8E A042  LDS  #5A042
          *
0003 7E 038E R      JMP  PASS1
          *
          *
          * MNTAB MNEMONIC TABLE *
          *
0006 41      MNTAB FCC  'ABA'
0009 106A  R      FDB  ADDR9
000B 1B      FCB  $1B
000C 41      FCC  'ADC'
000F 0E00  R      FDB  ADDR1
0011 09      FCB  $09
0012 41      FCC  'ADD'
0015 0E00  R      FDB  ADDR1
0017 0B      FCB  $0B
0018 41      FCC  'AND'
001B 0E00  R      FDB  ADDR1
001D 04      FCB  $04
001E 41      FCC  'ASL'
0021 0EF9  R      FDB  ADDR3
0023 0B      FCB  $0B
0024 41      FCC  'ASR'
0027 0EF9  R      FDB  ADDR3
0029 07      FCB  $07
002A 42      FCC  'BCC'
002D 1019  R      FDB  ADDR3
002F 24      FCB  $24
0030 42      FCC  'BCS'
0033 1019  R      FDB  ADDR8
0035 25      FCB  $25
0036 42      FCC  'BEQ'
0039 1019  R      FDB  ADDR8
003B 27      FCB  $27
003C 42      FCC  'BGE'
003F 1019  R      FDB  ADDR8
0041 2C      FCB  $2C
0042 42      FCC  'BGT'
0045 1019  R      FDB  ADDR8
0047 2E      FCB  $2E
0048 42      FCC  'BHI'
004B 1019  R      FDB  ADDR8
004D 22      FCB  $22
004E 42      FCC  'BIL'
0051 0E00  R      FDB  ADDR1
0053 05      FCB  $05
0054 42      FCC  'BLE'
0057 1019  R      FDB  ADDR8
0059 2F      FCB  $2F
005A 42      FCC  'BLS'
005D 1019  R      FDB  ADDR8
005F 23      FCB  $23
0060 42      FCC  'BLT'
0063 1019  R      FDB  ADDR8
0065 2D      FCB  $2D
0066 42      FCC  'BMI'
0069 1019  R      FDB  ADDR8
006B 2B      FCB  $2B
006C 42      FCC  'BNE'
006F 1019  R      FDB  ADDR8
0071 26      FCB  $26
0072 42      FCC  'BPL'
0075 1019  R      FDB  ADDR8
0077 2A      FCB  $2A
0078 42      FCC  'BRA'
007B 1019  R      FDB  ADDR8

```

```

007D 20      FCB  $20
007E 42      FCC  'BSR'
0081 1019  R      FDB  ADDR8
0083 8D      FCB  $8D
0084 42      FCC  'BVC'
0087 1019  R      FDB  ADDR8
0089 23      FCB  $23
008A 42      FCC  'BVS'
008D 1019  R      FDB  ADDR3
008F 29      FCB  $29
0090 43      FCC  'CBA'
0093 106A  R      FDB  ADDR9
0095 11      FCB  $11
0096 43      FCC  'CLC'
0099 106A  R      FDB  ADDR9
009B 0C      FCB  $0C
009C 43      FCC  'CLI'
009F 106A  R      FDB  ADDR9
00A1 0E      FCB  $0E
00A2 43      FCC  'CLR'
00A5 0EF9  R      FDB  ADDR3
00A7 0F      FCB  $0F
00A8 43      FCC  'CLV'
00AB 106A  R      FDB  ADDR9
00AD 0A      FCB  $0A
00AE 43      FCC  'CMN'
00B1 11D4  R      FDB  POCMN
00B3 FF      FCB  $FF
00B4 43      FCC  'CMP'
00B7 0E00  R      FDB  ADDR1
00B9 01      FCB  $01
00BA 43      FCC  'COM'
00BD 0EF9  R      FDB  ADDR3
00BF 03      FCB  $03
00C0 43      FCC  'CPX'
00C3 0F68  R      FDB  ADDR5
00C5 8C      FCB  $8C
00C6 44      FCC  'DAA'
00C9 106A  R      FDB  ADDR9
00CB 19      FCB  $19
00CC 44      FCC  'DEC'
00CF 0EF9  R      FDB  ADDR3
00D1 0A      FCB  $0A
00D2 44      FCC  'DES'
00D5 106A  R      FDB  ADDR9
00D7 34      FCB  $34
00D8 44      FCC  'DEX'
00DB 106A  R      FDB  ADDR9
00DD 07      FCB  $07
00DE 45      FCC  'END'
00E1 124E  R      FDB  POEND
00E3 FF      FCB  $FF
00E4 45      FCC  'ENT'
00E7 13D8  R      FDB  POENT
00E9 FF      FCB  $FF
00EA 45      FCC  'EOR'
00ED 0E00  R      FDB  ADDR1
00EF 03      FCB  $03
00F0 45      FCC  'EQU'
00F3 1451  R      FDB  POEQU
00F5 FF      FCB  $FF
00F6 45      FCC  'EX1'
00F9 14AF  R      FDB  POEX1
00FB FF      FCB  $FF
00FC 46      FCC  'FCB'
00FF 1511  R      FDB  POFGB
0101 FF      FCB  $FF
0102 46      FCC  'FCC'
0105 1546  R      FDB  POFCC
0107 FF      FCB  $FF
0108 46      FCC  'FDB'
010B 159D  R      FDB  POFDB
010D FF      FCB  $FF
010E 49      FCC  'IF'
0111 15F1  R      FDB  POIF
0113 FF      FCB  $FF

```

0114 49		FCC	'INC'
0117 0EF9	R	FDB	ADDR3
0119 0C		FCB	\$0C
011A 49		FCC	'INS'
011D 106A	R	FDB	ADDR9
011F 31		FCB	\$31
0120 49		FCC	'INX'
0123 106A	R	FDB	ADDR9
0125 08		FCB	\$08
0126 4A		FCC	'JMP'
0129 0FE6	R	FDB	ADDR7
012B 6E		FCB	\$6E
012C 4A		FCC	'JSR'
012F 0FE6	R	FDB	ADDR7
0131 AD		FCB	\$AD
0132 4C		FCC	'LDA'
0135 0E00	R	FDB	ADDR1
0137 06		FCB	\$06
0138 4C		FCC	'LDS'
013B 0F68	R	FDB	ADDR5
013D 8E		FCB	\$8E
013E 4C		FCC	'LDX'
0141 0F68	R	FDB	ADDR5
0143 CE		FCB	\$CE
0144 4C		FCC	'LSR'
0147 0EF9	R	FDB	ADDR3
0149 04		FCB	\$04
014A 4D		FCC	'MAC'
014D 1631	R	FDB	POMAC
014F FF		FCB	\$FF
0150 4E		FCC	'NAM'
0153 1726	R	FDB	PONAM
0155 FF		FCB	\$FF
0156 4E		FCC	'NEG'
0159 0EF9	R	FDB	ADDR3
015B 00		FCB	\$00
015C 4E		FCC	'NIF'
015F 175B	R	FDB	PONIF
0161 FF		FCB	\$FF
0162 4E		FCC	'NOP'
0165 106A	R	FDB	ADDR9
0167 02		FCB	\$02
0168 4F		FCC	'ORA'
016B JE00	R	FDB	ADDR1
016D 0A		FCB	\$0A
016E 50		FCC	'PAG'
0171 17A0	R	FDB	POPAG
0173 FF		FCB	\$FF
0174 50		FCC	'PSH'
0177 0F48	R	FDB	ADDR4
0179 36		FCB	\$36
017A 50		FCC	'PUL'
017D 0F48	R	FDB	ADDR4
017F 32		FCB	\$32
0180 52		FCC	'RMB'
0183 17C1	R	FDB	PORMB
0185 FF		FCB	\$FF
0186 52		FCC	'ROL'
0189 0EF9	R	FDB	ADDR3
018B 09		FCB	\$09
018C 52		FCC	'ROR'
018F 0EF9	R	FDB	ADDR3
0191 06		FCB	\$06
0192 52		FCC	'RTI'
0195 106A	R	FDB	ADDR9
0197 3B		FCB	\$3B
0198 52		FCC	'RTS'
019B 106A	R	FDB	ADDR9
019D 39		FCB	\$39
019E 53		FCC	'SBA'
01A1 106A	R	FDB	ADDR9
01A3 10		FCB	\$10
01A4 53		FCC	'SBC'
01A7 0E00	R	FDB	ADDR1
01A9 02		FCB	\$02
01AA 53		FCC	'SEC'
01AD 106A	R	FDB	ADDR9
01AF 00		FCB	\$00
01B0 53		FCC	'SEI'
01B3 106A	R	FDB	ADDR9
01B5 0F		FCB	\$0F
01B6 53		FCC	'SEV'
01B9 106A	R	FDB	ADDR9
01BB 08		FCB	\$08
01BC 53		FCC	'STA'
01BF 0E91	R	FDB	ADDR2
01C1 07		FCB	\$07
01C2 53		FCC	'STS'
01C5 0FDA	R	FDB	ADDR6
01C7 8F		FCB	\$8F
01C8 53		FCC	'STX'
01CB 0FDA	R	FDB	ADDR6

01CD CF		FCB	\$CF
01CE 53		FCC	'SUB'
01D1 0E00	R	FDB	ADDR1
01D3 00		FCB	\$00
01D4 53		FCC	'SWI'
01D7 106A	R	FDB	ADDR9
01D9 3F		FCB	\$3F
01DA 54		FCC	'TAB'
01DD 106A	R	FDB	ADDR9
01DF 16		FCB	\$16
01E0 54		FCC	'TAP'
01E3 106A	R	FDB	ADDR9
01E5 06		FCB	\$06
01E6 54		FCC	'TBA'
01E9 106A	R	FDB	ADDR9
01EB 17		FCB	\$17
01EC 54		FCC	'TPA'
01EF 106A	R	FDB	ADDR9
01F1 07		FCB	\$07
01F2 54		FCC	'TSI'
01F5 0EF9	R	FDB	ADDR3
01F7 0D		FCB	\$0D
01F8 54		FCC	'TSX'
01FB 106A	R	FDB	ADDR9
01FD 30		FCB	\$30
01FE 54		FCC	'TXS'
0201 106A	R	FDB	ADDR9
0203 35		FCB	\$35
0204 57		FCC	'WAI'
0207 106A	R	FDB	ADDR9
0209 3E		FCB	\$3E

* CHARACTER TABLE
*

	CHRTAB	FCB	\$00	SPACE
020A 00		FCB	\$00	!
020B 00		FCB	\$00	"
020C 00		FCB	\$00	#
020D 04		FCB	\$04	\$
020E 04		FCB	\$04	%
020F 04		FCB	\$04	&
0210 00		FCB	\$00	'
0211 04		FCB	\$04	(
0212 00		FCB	\$00	)
0213 00		FCB	\$00	*
0214 24		FCB	\$24	+
0215 24		FCB	\$24	,
0216 04		FCB	\$04	-
0217 24		FCB	\$24	.
0218 80		FCB	\$80	/
0219 24		FCB	\$24	0
021A 42		FCB	\$42	1
021B 42		FCB	\$42	2
021C 42		FCB	\$42	3
021D 42		FCB	\$42	4
021E 42		FCB	\$42	5
021F 42		FCB	\$42	6
0220 42		FCB	\$42	7
0221 42		FCB	\$42	8
0222 42		FCB	\$42	9
0223 42		FCB	\$42	:
0224 00		FCB	\$00	;
0225 00		FCB	\$00	<
0226 00		FCB	\$00	=
0227 00		FCB	\$00	>
0228 00		FCB	\$00	?
0229 00		FCB	\$00	@
022A 80		FCB	\$80	A
022B 83		FCB	\$83	B
022C 83		FCB	\$83	C
022D 82		FCB	\$82	D
022E 82		FCB	\$82	E
022F 82		FCB	\$82	F
0230 82		FCB	\$82	G
0231 80		FCB	\$80	H
0232 80		FCB	\$80	I
0233 80		FCB	\$80	J
0234 80		FCB	\$80	K
0235 80		FCB	\$80	L
0236 80		FCB	\$80	M
0237 80		FCB	\$80	N
0238 80		FCB	\$80	O
0239 80		FCB	\$80	P
023A 80		FCB	\$80	Q
023B 80		FCB	\$80	R
023C 80		FCB	\$80	S
023D 80		FCB	\$80	T
023E 80		FCB	\$80	U
023F 80		FCB	\$80	V
0240 80		FCB	\$80	W
0241 80		FCB	\$80	X
0242 81		FCB	\$81	Y
0243 80		FCB	\$80	

```

0244 80          FCB $80 Z
0245 00          FCB $00 [
0246 00          FCB $00 \
0247 00          FCB $00 ]
0248 00          FCB $00 CAROT
0249 00          FCB $00 UNDERLINE

```

* MAIN PROGRAM LOOP *

```

*
*
024A 7E 0000 X    EXT TABLES
024D 7E 0000 X    EXT UPDAIE
0250 7E 0000 X    EXT MONIOR
0253 7E 0000 X    EXT GETB
0256 7E 0000 X    EXT OUTB
0259 7E 0000 X    EXT WREOF
025C 7E 0000 X    EXT INITIO
025F 7E 0000 X    EXT RESTR
*
0262 1861 N      ENT PDATAI
0262 0388 N      ENI INEE
0262 13BE N      ENT CRLF
*
0262 0002      MACTBL RMB 2 MACRO TABLE
0264 0002      MACEND RMB 2 MACRO TABLE END
0266 0002      MACSTK RMB 2 MACRO STACK
0268 0002      SYMTAB RMB 2 SYMBOL TABLE
026A 0002      NSYM  RMB 2 NUMBER OF SYMBOLS
026C 0002      SYMEND RMB 2 SYMTAB END
*
026E 0001      OPTNS  RMB 1  OPTIONS
026F 0002      LNUM   RMB 2  LINE NUMBER
0271 0002      I$TPH  RMB 2  PHASING ERROR CHECK LOC.
0273 0002      LC     RMB 2  LOCATION COUNTER
0275 0001      PASS   RMB 1  PASS; 0=1, FF=2
0276 0001      LBFLG  RMB 1  0=NO LABEL
0277 0001      RELFLG RMB 1  RELOCATION FLAG 00=NO,FF=YES
0278 0001      CMNFLG RMB 1  COMMON FLAG      "
0279 0001      EXTFLG RMB 1  EXTERNAL FLAG    "
027A 0001      ENIFLG RMB 1  ENTRY FLAG      "
027B 0002      DESCRA RMB 2  DESCRIPTOR ADDRESS
027D 0001      DESCRC RMB 1  DESCRIPTOR COUNT
027E 0002      CUCHAR RMB 2  CURRENT CHAR ADDRESS
0280 0002      CULINE RMB 2  CURRENT LINE ADDRESS
0282 0002      SYMPTR RMB 2  SYMTAB POINTER
0284 0001      LCN     RMB 1  # BYTES IN AN INSTRUCTION
0285 0002      LSAVE  RMB 2  LC SAVE LOCATION
0287 0001      LCOUNT RMB 1  # LINES ON A PAGE
0288 0002      ECOUNT  RMB 2  ERROR COUNT
028A 0002      MSTKPT  RMB 2  MACRO STACK POINTER
028C 0002      STKSAV  RMB 2  MACRO STACK POINTER SAVE
028E 0002      MXSAV1  RMB 2  MACRO TEMP SAV
0290 0002      MXSAV2  RMB 2  MACRO TEMP SAV
0292 0048      MACLIN  RMB 72 MACRO EXPANSION LINE AREA
02DA 0032      MACPAR  RMB 50 MACRO PARAMETER AREA
030C 0001      MACFLG  RMB 1  MACRO MODE; 00=NORMAL, FF=MACRO
030D 0002      MACPTR  RMB 2  POINTER TO MACTABLE
030F 0002      MACSAV  RMB 2  MACRO X-REG SAVE AREA
0311 0002      MCLPTR  RMB 2  MACLIN POINTER
0313 0002      CMNLC   RMB 2  COMMON_BLOCK LC
0315 0050      INLINE  RMB 80 INPUT LINE
0365 0010      RMB 16
0375 0375      R IFSTK EQU * IF STACK
0375 0002      @IFSTK RMB 2 IF STACK POINTER
0377 0001      IFFLG  RMB 1 IF FLAG 00=NO ASSEMBLY; FF=ASSEMBLY
*
0378 45      OPTMSG FCC 'ENTER OPTIONS:
0387 04      FCB 4
*
*
0388 7E E1AC INEEE JMP SEIAC INPUT A CHAR FROM TTY
038B 7E E1D1 OUTEEE JMP SEIDI
*
*
*
* PASS 1 IS ENTRY POINT TO ASSEMBLER
*
033E 7F 0275 R PASS1 CLR PASS PASS=1
0391 CE 0378 R OPIN  LDX #OPTMSG
0394 BD 1861 R      JSR PDATAI
*
0397 7F 026E R      CLR OPTNS
039A 73 026E R      COM OPTNS NL,NO,NM,NS
*
039D BD 0388 R OPINI JSR INEEE GET OPTION
03A0 81 0D      CMP A #50D CR ?
03A2 27 26      BEQ OPIN3 YES
*
03A4 81 4C      CMP A #'L LIST?

```

03A6 26 04	*	BNE **6	NO
03A6 86 70		LDA A #S70	YES
03AA 20 16	*	BRA OPIN2	
03AC 81 4F		CMP A #'0	OBJECT?
03AE 26 04	*	BNE **6	NO
03B0 86 80		LDA A #S80	YES
03B2 20 0E	*	BRA OPIN2	
03B4 81 53		CMP A #'S	SYMBOL TABLE?
03B6 26 04	*	BNE **6	NO
03B8 86 D0		LDA A #SD0	YES
03BA 20 06	*	BRA OPIN2	
03BC 81 4D		CMP A #'M	MACRO EXPANSION LISTING?
03BE 26 D0	*	BNE OPIN1	NO
03C0 86 E0	*	LDA A #SE0	YES
03C2 B4 026E R	OPIN2	AND A OPINS	TURN OFF "NOT" BIT
03C5 B7 026E R		STA A OPINS	
03C8 20 D3	*	BRA OPIN1	GET ANOTHER OPTION
03CA BD 13BE R	OPIN3	JSR CRLF	
	*		
	*	CONFIGURE TABLES	
	*		
03CD FE 024B R		LDX TABLES+1	
03D0 EE 00		LDX 0,X	GET START OF TABLES
03D2 FF 0262 R	*	STX MACTBL	INIT MACTBL
03D5 FF 07E3 R		STX PSING1	
03D8 CE 0800		LDX #S0800	
03DB FF 07E5 R		STX PSING2	
03DE CE 07E3 R		LDX #PSING1	
03E1 BD 0BEC R		JSR ADD16	
03E4 FE 07E3 R		LDX PSING1	
03E7 FF 0264 R	*	STX MACEND	INIT MACEND
03EA CE 0100		LDX #S0100	
03ED FF 07E5 R		STX PSING2	
03F0 CE 07E3 R		LDX #PSING1	
03F3 BD 0BEC R		JSR ADD16	
03F6 FE 07E3 R		LDX PSING1	
03F9 FF 0266 R	*	STX MACSTK	INIT MACSTK
03FC 08		INX	
03FD FF 0268 R	*	STX SYMTAB	INIT SYMTAB
0400 FF 07E5 R		STX PSING2	
0403 CE 3FFF R		LDX #ASM+S3FFF 16K	
0406 FF 07E3 R		STX PSING1	
0409 CE 07E3 R		LDX #PSING1	
040C BD 0BFD R		JSR SUB16	
040F B6 07E3 R		LDA A PSING1	
0412 F6 07E4 R		LDA B PSING1+1	
0415 CE 0009		LDX #0009	
0418 FF 07E5 R		STX PSING2	
041B CE 07E5 R		LDX #PSING2	
041E BD 0BA1 R		JSR DIV16	
0421 B7 026A R		STA A NSYM	
0424 F7 026B R	*	STA B NSYM+1	INIT NSYM
0427 CE 0009		LDX #0009	
042A FF 07E3 R		STX PSING1	
042D CE 07E3 R		LDX #PSING1	
0430 BD 0B7D R		JSR MPY16	
0433 B7 07E3 R		STA A PSING1	
0436 F7 07E4 R		STA B PSING1+1	
0439 FE 026B R		LDX SYMTAB	
043C FF 07E5 R		STX PSING2	
043F CE 07E3 R		LDX #PSING1	
0442 BD 0BEC R		JSR ADD16	
0445 FE 07E3 R		LDX PSING1	
0448 FF 026C R	*	STX SYMEND	
044B 86 20	*	LDA A #S20	BLANKS TO SYMTAB
044D FE 0268 R		LDX SYMTAB	POINT TO SYMTAB
0450 A7 00		STA A 0,X	BLANK LOCATION
0452 08		INX	BUMP POINTER
0453 BC 026C R		CPX SYMEND	ALL DONE ?
0456 26 F8	*	BNE *-6	NO
0458 CE 0000		LDX #S0000	
045B FF 0271 R		STX TSIPH	CLEAR TSIPH
045E FF 0288 R		STX ECOUNT	CLEAR ECOUNT
0461 CE 0000		LDX #S0000	
0464 FF 0313 R	*	STX CMNLC	INIT COMMON LC

0467	BD	1089	R	PASS2	JSR	ADRINT	CLEAR FLAGS
046A	7F	0287	R		CLR	LCOUNT	LCOUNT:=0
046D	FE	0262	R		LDX	MACTBL	INIT MACPTR
0470	FF	030D	R		STX	MACPTR	
0473	7F	030C	R		CLR	MACFLG	MODE:= NON-MACRO
0476	FE	0266	R		LDX	MACSTK	INIT MACRO STACK POINTER
0479	FF	028A	R		STX	MSTKPT	
047C	86	FF			LDA	A #FFF	
047E	B7	0377	R		STA	A IFFLG	INIT TO ASSEMBLE
0481	CE	0375	R		LDX	#IFSTK	
0484	FF	0375	R		STX	#IFSTK	INIT IFSTK
0487	CE	0000			LDX	#\$0000	
048A	FF	0273	R		STX	LC	INIT LC
048D	FF	026F	R		STX	LNUM	INIT LNUM
★							
★ MAIN IS THE DRIVER SECTION OR TOP LEVEL							
★ OF THE ASSEMBLER							
★							
0490	BD	0568	R	MAIN1	JSR	RDLINE	GET A LINE OF SOURCE
0493	7F	0276	R		CLR	LBFLG	SET FLAG TO NO LABEL
0496	FE	026F	R		LDX	LNUM	
0499	08				INX		
049A	FF	026F	R		STX	LNUM	BUMP LNUM
049D	FE	0280	R		LDX	CULINE	POINT TO LINE
04A0	A6	00			LDA	A 0,X	GET COL 1
04A2	81	2A			CMP	A #\$2A	COMMENT?
04A4	26	08			BNE	MAIN3	NO
★							
04A6	BD	1089	R	MAIN1A	JSR	ADRINT	CLEAR PRINT FLAGS
04A9	BD	0C0E	R		JSR	PRINTL	PRINT THE LINE
04AC	20	E2			BRA	MAIN1	
★							
04AE	7D	0377	R	MAIN3	TST	IFFLG	ASSEMBLING?
04B1	26	22			BNE	MAIN3C	YES
★							
04B3	81	20			CMP	A #\$20	COL 1 BLANK?
04B5	27	03			BEQ	MAIN3A	YES
★							
04B7	BD	06F6	R		JSR	NXTOK	SCAN OVER LABEL
04BA	BD	06F6	R	MAIN3A	JSR	NXTOK	GET MNEMONIC
04BD	B6	027D	R		LDA	A DESCRC	GET COUNT
04C0	81	03			CMP	A #3	<= 3?
04C2	22	E2			BHI	MAIN1A	NO
★							
04C4	BD	0940	R		JSR	MNLKP	SEARCH MNTAB
04C7	8C	15F1	R		CPX	#POIF	IF ?
04CA	27	07			BEQ	MAIN3B	YES
★							
04CC	8C	175B	R		CPX	#PONIF	NIF?
04CF	27	02			BEQ	MAIN3B	YES
★							
04D1	20	D3			BRA	MAIN1A	NEITHER
★							
04D3	6E	00		MAIN3B	JMP	0,X	GO TO IF OR NIF PROCESSING ROUTINE
★							
04D5	81	20		MAIN3C	CMP	A #\$20	COL 1 BLANK?
04D7	27	1D			BEQ	MAIN5	YES
★							
04D9	BD	06F6	R		JSR	NXTOK	GET LABEL
04DC	C1	01			CMP	B #\$01	OK?
04DE	27	0B			BEQ	MAIN4	YES
★							
04E0	CE	0205			LDX	#\$0205	ERROR
04E3	BD	0D8B	R		JSR	PRINTE	
04E6	BD	0C0E	R		JSR	PRINTL	PRINT LINE
04E9	20	A5			BRA	MAIN1	
★							
04EB	7C	0276	R	MAIN4	INC	LBFLG	SET LABEL FLAG
04EE	7D	0275	R		TST	PASS	PASS?
04F1	26	03			BNE	MAIN5	PASS2
★							
04F3	BD	07F8	R		JSR	STOSYM	STORE LABEL IN SYMTAB
★							
04F6	BD	06F6	R	MAIN5	JSR	NXTOK	GET MNEMONIC
04F9	C1	01			CMP	B #\$01	OK?
04FB	27	0B			BEQ	MAIN7	YES
★							
04FD	CE	0202		MAIN6	LDX	#\$0202	ERROR
0500	BD	0D8B	R		JSR	PRINTE	
0503	BD	0C0E	R		JSR	PRINTL	PRINT LINE
0506	20	88			BRA	MAIN1	
★							
0508	BD	0940	R	MAIN7	JSR	MNLKP	SEARCH MNTAB
050B	81	00			CMP	A #\$00	IN MNTAB?
050D	27	2D			BEQ	MAIN9	YES
★							
050F	BD	0857	R		JSR	LKPSYM	MACRO NAME?
0512	C1	FF			CMP	B #\$FF	IN SYMTAB?
0514	27	28			BEQ	MAIN8	NO,ERROR
★							
0516	C5	20			BIT	B #\$20	MACRO NAME?

```

0510 27 24          BEQ  MAIN8      NO,ERROR
*
*
051A 7D 030C R      1ST  MACFLG      MACRO MODE?
051D 27 08          BEQ  MAIN7A      NO
*
* PUSH PRESENT MACRO ONTO MACSTACK
*
051F BD 0637 R      JSR  MACPSH
0522 7D 030C R      1ST  MACFLG      ERRORS?
0525 27 20          BEQ  MAIN13      YES
*
0527 FF 030D R      MAIN7A STX  MACPTR      SAVE MACRO LOC IN MACTBL
052A BD 0C0E R      JSR  PRINTL
052D 7C 030C R      INC  MACFLG      MODE:=MACRO
*
0530 BD 06F6 R      JSR  NXTOK      PARMS?
0533 C1 0D          CMP  B  #SOD
0535 26 13          BNE  MAIN12      YES, SAVE THEM
*
0537 F7 02DA R      SIA  B  MACPAR      NO, CR TO MACPAR
053A 2J 0B          BRA  MAIN13
*
053C 6E 00          MAIN9  JMP  O,X      GO TO ROUTINE
*
053E CE 0207          MAIN8  LDX  #S0207  ERROR
0541 BD 00BB R      JSR  PRINTE      PRINT IT
0544 BD 0C0E R      MAIN10 JSR  PRINTL      PRINT LINE
0547 7E 0490 R      MAIN13 JMP  MAIN1  PROCESS NEXT LINE
*
* MOVE PARMS ON MACRO CALL TO MACPAR
*
054A CE 02DA R      MAIN12 LDX  #MACPAR
054D FF 030F R      SIX  MACSAV      INIT POINTER
*
0550 FE 027B R      MAIN11 LDX  DESCRA      POINT TO PARMS
0553 A6 00          LDA  A  O,X      GET A CHAR
0555 08            INX
0556 FF 027B R      SIX  DESCRA      SAVE POINTER
*
0559 FE 030F R      LDX  MACSAV      GET POINTER
055C A7 00          SIA  A  O,X      MOVE CHAR
055E 08            INX
055F FF 030F R      SIX  MACSAV      SAVE POINTER
*
0562 81 0D          CMP  A  #SOD      EOL?
0564 26 EA          BNE  MAIN11      NO
*
0566 20 DF          BRA  MAIN13      YES
*
* GET A LINE OF SOURCE FROM INBUF *
* RETURNS ADDRESS OF LINE IN CULINE *
* CUCHAR:=ADDRESS OF FIRST CHARACTER*
*
0568 7D 030C R      RDLIN 1ST  MACFLG      MACRO MODE?
056B 27 09          BEQ  RDLIN      NO
*
056D BD 05A5 R      JSR  RDMAC      EXPAND MACRO
*
0570 7D 030C R      1ST  MACFLG      MACRO FULLY EXPANDED?
0573 27 01          BEQ  RDLIN      YES
*
0575 39            RTS
*
0576 CE 0315 R      RDLIN 1 LDX  #INLINE
0579 FF 027E R      STX  CUCHAR
057C FF 0280 R      STX  CULINE
*
057F BD 0253 R      RDL1  JSR  GETB
0582 24 06          BCC  RDL1A
*
0584 8E A042          LDS  #A042      FLUSH STACK, EOF
0587 7E 1254 R      JMP  POENDO
*
058A 81 0A          RDL1A  CMP  A  #S0A      LF?
058C 27 F1          BEQ  RDL1      YES
*
058E 81 00          CMP  A  #S00      NULLS?
0590 27 ED          BEQ  RDL1      YES
*
0592 8C 0364 R      CPX  #INLINE+79 LINE TO LONG?
0595 27 05          BEQ  RDL2      YES
*
0597 A7 00          STA  A  O,X      STORE CHARACTER
0599 08            INX
059A 2J 04          BRA  RDL3
*
059C C6 0D          RDL2  LDA  B  #SOD      TRUNCATE LINE
059E E7 00          STA  B  O,X
*
05A0 81 0D          RDL3  CMP  A  #SOD      CR?
05A2 26 DB          BNE  RDL1      NO
*

```

```

05A4 39          RTS
                * RDMAC: EXPAND MACRO CALLS
                *
05A5 FE 030D R RDMAC LDX MACPTR
05A8 A6 00          LDA A 0,X      GET CHAR
05AA 81 17          CMP A #S17     E1B?
05AC 26 0B          BNE RDMAC1     NO
                *
05AE 7A 030C R      DEC MACFLG     DEC MODE COUNT
05B1 27 05          BEQ RDMAC0     NO MORE MACROS
                *
                * PULL UP LAST MACRO STACKED
                *
05B3 BD 069B R      JSR MACPUL
05B6 20 ED          BRA RDMAC
                *
05B8 39          RDMAC0 RTS
                *
05B9 CE 0292 R RDMAC1 LDX #MACLIN  POINT TO MACRO EXPAND AREA
05BC FF 0280 R      STX CULINE     INIT
05BF FF 027E R      STX CUCHAR     INIT
05C2 FF 0311 R      STX MCLPTR     INIT
                *
05C5 FE 030D R RDMAC2 LDX MACPTR    POINT TO MACRO DEF
05C8 A6 00          LDA A 0,X
05CA 08          INX
                *
05CB FF 030D R      STX MACPTR
05CE 81 20          CMP A #8       MACRO PARM?
05D0 27 13          BEQ RDMAC3     YES
                *
05D2 FE 0311 R      LDX MCLPTR     POINT TO MACLIN
05D5 A7 00          STA A 0,X      MOVE CHAR TO MACLIN
05D7 08          INX
05D8 FF 0311 R      STX MCLPTR     SAVE POINTER
05DB 8C 02D9 R      CPX #MACLIN+71 OVERFLOW?
05DE 27 4B          BEQ RDMERR     YES
                *
05E0 81 00          CMP A #S0D     EOL?
05E2 26 E1          BNE RDMAC2     NO
05E4 39          RTS              ALL DONE
                *
                * SUBSTITUTE POSITIONAL PARMS
                *
05E5 E6 00          RDMAC3 LDA B 0,X  GET POSITIONAL # OF PARM
05E7 C0 2F          SUB B #S2F      CONVERT TO BINARY
05E9 08          INX              SKIP OVER POS#
05EA FF 030D R      STX MACPTR
05ED CE 02DA R      LDX #MACPAR    POINT TO PARMS FROM CALL
                *
                * SCAN OVER PARMS
                *
05F0 FF 030F R RDMAC6 STX MACSAV    SAVE
                *
05F3 A6 00          RDMAC4 LDA A 0,X  GET A CHAR
05F5 08          INX
05F6 81 2C          CMP A #',      END OF PARM?
05F8 27 04          BEQ RDMAC7     YES
                *
05FA 81 00          CMP A #S0D     EOL?
05FC 26 F5          BNE RDMAC4     NO
                *
05FE 5A          RDMAC7 DEC B        FOUND PARM?
05FF 26 EF          BNE RDMAC6     NO
                *
0601 FE 030F R      LDX MACSAV     POINT TO PARM
0604 A6 00          LDA A 0,X      FOUND PARM, GET CHAR
0606 08          INX
0607 FF 030F R      STX MACSAV     SAVE POINTER
                *
060A FE 0311 R RDMAC5 LDX MCLPTR    POINT TO MACLIN
060D A7 00          STA A 0,X      MOVE CHAR
060F 08          INX
0610 FF 0311 R      STX MCLPTR     SAVE POINTER
0613 8C 02D9 R      CPX #MACLIN+71 OVERFLOW?
0616 27 13          BEQ RDMERR     YES
                *
                *
0618 FE 030F R      LDX MACSAV     POINT TO MACPAR
061B A6 00          LDA A 0,X      GET NEXT CHAR
061D 08          INX
061E FF 030F R      STX MACSAV     SAVE
0621 81 2C          CMP A #',      END OF PARM?
0623 27 A0          BEQ RDMAC2     YES
                *
0625 81 00          CMP A #S0D     EOL?
0627 26 E1          BNE RDMAC5     NO
                *
0629 20 9A          BRA RDMAC2
                *
062B 86 0D          RDMERR LDA A #S0D  END LINE
062D A7 00          STA A 0,X

```

```

062F CE 0230      LDX #0230      ERROR MESSAGE
0632 BD 00BB R    JSR PRINTC
0635 20 8E        BRA RDMAC2

* PUSH A MACRO ONTO THE MACRO STACK
*
0637 FF 028E R    MACPSH STX MXSAV1      SAVE X-REG
063A BF 028C R    STS STKSAV      SAVE STACK POINTER
063D BE 023A R    LDS MSTKPT      LOAD MACRO STACK POINTER

*
* PUSH MCLPTR,MACSAV,MACPTR ONTO STACK
*
0640 CE 0312 R    LDX #MCLPTR+1
0643 C6 06        LDA B #6

*
0645 A6 00        MPSH1 LDA A 0,X      GET A BYTE
0647 09          DEX
0648 FF 0290 R    STX MXSAV2      SAVE POINTER
064B 30          TSX              X*=STKPTR+1
064C 09          DEX
064D BC 0264 R    CPX MACEND      END OF STACK?
0650 27 33        BEQ MPSH5      YES,ERROR

*
0652 FE 0290 R    LDX MXSAV2      RESTORE POINTER
0655 36          PSH A            PUSH A BYTE ONTO THE STACK
0656 5A          DEC B            ALL DONE?
0657 26 EC        BNE MPSH1      NO

*
* PUSH MACPAR IN REVERSE ORDER
*
0659 CE 02DA R    LDX #MACPAR

*
065C A6 00        MPSH2 LDA A 0,X      FIND EOL
065E 81 0D        CMP A #00D      EOL?
0660 27 03        BEQ MPSH3      YES

*
0662 08          INX
0663 20 F7        BRA MPSH2

*
0665 A6 00        MPSH3 LDA A 0,X      GET A BYTE
0667 FF 0290 R    STX MXSAV2      SAVE PTR
066A 30          TSX              X*=STKPTR+1
066B 09          DEX
066C BC 0264 R    CPX MACEND      END OF STACK?
066F 27 14        BEQ MPSH5      YES,ERROR

*
0671 FE 0290 R    LDX MXSAV2      RESTORE POINTER
0674 36          PSH A            PUSH A BYTE ONTO THE STACK
0675 09          DEX              POINT TO NEXT LEFT CHAR
0676 8C 02D9 R    CPX #MACPAR-1    ALL DONE?
0679 26 EA        BNE MPSH3      NO

*
067B BF 028A R    STS MSTKPT      SAVE STACK POINTER
067E BE 028C R    LDS STKSAV      RESTORE STACK
0681 FE 028E R    LDX MXSAV1      RESTORE X-REG
0684 39          RTS

*
* MACRO NESTING OVERFLOW ERROR
*
0685 BE 0266 R    MPSH5 LDS MACSTK      FLUSH STACK
0688 BF 028A R    STS MSTKPT
068B BE 028C R    LDS STKSAV
068E CE 0251      LDX #0251      ERROR #
0691 BD 00BB R    JSR PRINTC
0694 FE 028E R    LDX MXSAV1      RESTORE X-REG
0697 7F 030C R    CLR MACFLG      GET OUT OF MACRO MODE
069A 39          RTS

*
* PULL A MACRO FROM THE MACRO STACK
*
069B FF 028E R    MACPUL STX MXSAV1      SAVE X-REG
069E BF 028C R    STS STKSAV      SAVE STACK POINTER
06A1 BE 028A R    LDS MSTKPT      LOAD MACRO STACK POINTER

*
* PULL MACPAR OFF OF THE MACRO STACK
*
06A4 CE 02DA R    LDX #MACPAR

*
06A7 32          MPUL1 PUL A            PULL A CHAR
06A8 A7 00        STA A 0,X          SAVE IN MACPAR
06AA 08          INX
06AB 81 0D        CMP A #00D      EOL?
06AD 26 F8        BNE MPUL1      NO

*
* PULL MACPTR,MACSAV,MCLPTR
*
06AF CE 030D R    LDX #MACPTR
06B2 C6 06        LDA B #6

*
06B4 32          MPUL2 PUL A            PUL A CHAR
06B5 A7 00        STA A 0,X          SAVE
06B7 08          INX
06B8 5A          DEC B
06B9 26 F9        BNE MPUL2      NOT DONE

```



```

*
06BB BF 028A R      STS MSTKPT      SAVE MACRO STACK PNTR
06BE BE 028C R      LDS STKSAV      RESTORE STACK POINTER
06C1 FE 028E R      LDX MXSAV1     RESTORE X-REG
06C4 39             RTS

* COMPARE TWO STRINGS *
* ON ENTRY (X) = A PARM LIST OF 5 BYTES :
*   A (STRING1)
*   A (STRING2)
*   COUNT OF # BYTES TO BE COMPARED
* ON RETURN IF CC Z IS SET THERE IS A MATCH
* EXAMPLE:
*           LDX #STRING1
*           JSR COMPAR
*           BEQ ----- MATCH
*
*   STRING1 RMB 2
*   STRING2 RMB 2
*   COUNT   RMB 1
*
06C5 36             COMPAR PSH A
06C6 37             PSH B
06C7 E6 04          LDA B 4,X      GET COUNT
06C9 FF 06F4 R      STX XSAV      SAVE PARM POINTER
06CC FE 06F4 R CMP1  LDX XSAV      GET PARM PTR
06CF EE 00          LDX 0,X      GET A (STRING1)
06D1 A6 00          LDA A 0,X     GET CHARACTER
06D3 FE 06F4 R      LDX XSAV      GET PTR
06D6 6C 01          INC 1,X      PTR SET TO NEXT
06D8 26 02          BNE CMP2      CHAR IN
06DA 6C 00          INC 0,X      STRING1
06DC FE 06F4 R CMP2  LDX XSAV      GET PARM PTR
06DF EE 02          LDX 2,X      GET A (STRING2)
06E1 A1 00          CMP A 0,X     COMPARE
06E3 26 0C          BNE CDONE     NOT EQUAL
06E5 FE 06F4 R      LDX XSAV      GET PARM POINTER
06E8 6C 03          INC 3,X      PTR SET TO NEXT
06EA 26 02          BNE CMP3      CHAR IN
06EC 6C 02          INC 2,X      STRING2
06EE 5A            CMP3 DEC B      DECREMENT COUNT
06EF 26 DB          BNE CMP1      TRY AGAIN
06F1 33            CDONE PUL B      DONE
06F2 32            PUL A
06F3 39            RTS

06F4 0002          XSAV RMB 2      PARM PTR SAVE AREA
* NEXT TOKEN ROUTINE *
* SCANS A LINE OF SOURCE CODE AND RETURNS
* THE NEXT TOKEN, CLASS & RC IN REGS A,B
* THE ADDRESS OF THE TOKEN IS RETURNED IN
* DESCRA AND THE # OF BYTES IN THE TOKEN IS
* RETURNED IN DESCRC.
* THE RC AND CLASS ARE:
*
* TYPE#   RC (B)   CLASS (A)
*
* NAME    01      02  SUBSTRINGS
* HEX     03      02
* DECIMAL 09      02
*
* #       23      04  DELIMITERS
* ,       2C      04
* '       27      04
*
* *       2A      24  ARITHMETIC
* /       2F      24
* +       2B      24
* -       2D      24
*
* A       41      01  A,B,X REGS
* B       42      01
* X       58      01
*
* CR      0D      0D  EOL
*
* ERROR   00      00  ERRORS
*
06F6 7F 027D R NXTOK CLR DESCRC
06F9 7C 027D R      INC DESCRC    DESCRC*=1
06FC FE 027E R NXTO LDX CUCHAR    POINT TO CURRENT CHAR
06FF FF 027B R      STX DESCRA    INIT DESCRA
0702 A6 00          LDA A 0,X     GET CHAR
0704 08            INX
0705 FF 027E R      STX CUCHAR    POINT TO NEXT CHAR
0708 81 20          CMP A #520    LESS THAN 20 HEX?
070A 27 F0          BEQ NXTO      BLANK, SKIP OVER
070C 22 06          BHI NXTO      >20
*
070E 81 0D          CMP A #50D    CR?
0710 26 47          BNE NXTER     NO, UNRECOG. CHAR

```

0712 16		TAB		YES, SET RC
0713 39		RTS		
*				
0714 81 5F	NXT1	CMP A	#55F	>5F?
0716 23 02		BLS	NXT3	NO
071b 20 3F		BRA	NXT1R	YES,UNRECOG. CHAR
*				
071A BD 07C3 R	NXT3	JSR	GCHRTB	GET BYTE FROM CHARTAB
071D 85 01		BIT A	#501	A,B,X REGS?
071F 27 13		BEQ	NXT4	NO
*				
0721 FE 027E R		LDX	CUCHAR	YES,CHECK NEXT CHAR
0724 E6 00		LDA B	0,X	
0726 C1 20		CMP B	#520	BLANK?
0728 27 04		BEQ	NXT3A	YES
072A C1 00		CMP B	#50D	EOL?
072C 26 0A		BNE	NX14A	NO GOT TO NSCAN
*				
072E FE 027B R	NXT3A	LDX	DESCRA	GET RC
0731 E6 00		LDA B	0,X	
0733 39		RTS		
*				
0734 85 80	NXT4	BIT A	#580	NAME?
0736 27 04		BEQ	NXT5	NO
0738 BD 0773 R	NXT4A	JSR	NSCAN	YES SCAN NAME STRING
073B 39		RTS		
*				
073C 85 40	NXT5	BIT A	#540	DECIMAL?
073E 27 04		BEQ	NXT6	NO
0740 BD 075C R		JSR	DSCAN	YES,SCAN DECIMAL STRING
0743 39		RTS		
*				
0744 85 20	NXT6	BIT A	#520	ARITHMETIC?
0746 26 E6		BNE	NXT3A	YES GET RC AND RTN
*				
0748 85 04		BIT A	#504	DELIMITERS?
074A 27 00		BEQ	NXT1R	NO,UNRECOG. CHAR
074C FE 027B R		LDX	DESCRA	GET CHAR
074F E6 00		LDA B	0,X	
0751 C1 24		CMP B	#524	S? (HEX)
0753 26 D9		BNE	NXT3A	NO,GET RC AND RTN
*				
0755 BD 0798 R		JSR	HSCAN	YES,SCAN HEX STRING
0758 39		RTS		
*				
0759 4F	NXT1R	CLR A		TROUBLE,SET RC,CLASS=00
075A 5F		CLR B		
075B 39		RTS		
* DSCAN SCAN DECIMAL STRING STOP AT				
* FIRST NON-DECIMAL CHAR				
*				
075C FE 027E R	DSCAN	LDX	CUCHAR	POINT TO NEXT CHAR
075F A6 00		LDA A	0,X	GET CHAR
0761 7C 027D R		INC	DESCRC	BUMP COUNT
0764 08		INX		
0765 FF 027E R		SIX	CUCHAR	POINT TO NEXT CHAR
076C BD 07C3 R		JSR	GCHRTB	GET BYTE IN CHARTAB
076B 85 40		BIT A	#540	DECIMAL?
076D 26 ED		BNE	DSCAN	YES CONTINUE SCAN
076F C6 09		LDA B	#509	
0771 20 43		BRA	ENDSCAN	RETURN
*				
*				
* NSCAN SCAN NAME STRING STOP AT				
* FIRST NON-ALPHANUMERIC CHAR				
*				
0773 FE 027E R	NSCAN	LDX	CUCHAR	POINT TO NEXT CHAR
0776 A6 00		LDA A	0,X	GET CHAR
0778 7C 027D R		INC	DESCRC	BUMP COUNT
077B 08		INX		
077C FF 027E R		SIX	CUCHAR	POINT TO NEXT CHAR
077F BD 07C3 R		JSR	GCHRTB	GET BYTE IN CHARTAB
0782 85 80		BIT A	#580	ALPHA?
0784 26 ED		BNE	NSCAN	YES CONTINUE SCAN
0786 85 40		BIT A	#540	NUMERIC?
0788 26 E9		BNE	NSCAN	YES CONTINUE SCAN
078A C6 07		LDA B	#507	NAME TOO LONG ?
078C F1 027D R		CMP B	DESCRC	
078F 24 03		BCC	NSCAN	NO
0791 F7 027D R		STA B	DESCRC	YES,TRUNCATE
0794 C6 01	NSCAN	LDA B	#501	LOAD RC
0796 20 1E		BRA	ENDSCAN	RETURN
*				
*				
* HSCAN SCAN HEX STRING STOP AT				
* FIRST NON-HEX CHAR				
*				
0798 7F 027D R	HSCAN	CLR	DESCRC	DESCRC:=0
079B FE 027E R		LDX	CUCHAR	POINT TO NEXT CHAR
079E FF 027B R		STX	DESCRA	INIT DESCRA
07A1 FE 027E R	HSCAN	LDX	CUCHAR	POINT TO NEXT CHAR

```

07A4 A6 00          LDA A 0,X      GET CHAR
07A6 7C 027D R      INC  DESCRC    BUMP COUNT
07A9 03             INX
07AA FF 027E R      STX  CUCHAR     POINT TO NEXT CHAR
07AD BD 07C3 R      JSR  GCHRTB     GET BYTE IN CHRTAB
07B0 85 02          BIT  A #502     HEX?
07B2 26 ED          BNE  HSCAN1     YES CONTINUE SCAN
07B4 C6 03          LDA B #503
*
07B6 7A 027D R      ENDSN DEC  DESCRC DESCRC*= CORRECT COUNT
07B9 FE 027E R      LDX  CUCHAR
07BC 09             DEX
07BD FF 027E R      STX  CUCHAR     CUCHAR*= CORRECT VALUE
07C0 86 02          LDA A #2        LOAD CLASS RC
07C2 39             RTS             ALL DONE

* GET BYTE IN CHRTAB INDEXED BY VALUE OF
* CHAR IN REG A
*
07C3 81 20          GCHRTB CMP A #520  VALID CHAR ?
07C5 25 16          BCS  GCHRTB      NO < 20
07C7 81 5F          CMP A #55F      VALID CHAR ?
07C9 22 12          BHI  GCHRTB      NO, > 5F
*
07CB 7F 07DF R      CLR  CHPTR      INIT PARM
07CE B7 07E0 R      STA A CHPTR+1    SAVE CHAR
07D1 CE 07DF R      LDX  #CHPTR      POINT TO PARM
07D4 BD 0BEC R      JSR  ADD16       ADD IN BASE OF CHARTAB
07D7 FE 07DF R      LDX  CHPTR      GET BYTE IN CHARTAB
07DA A6 00          LDA A 0,X
07DC 39             RTS
*
07DD 4F             GCHRTB CLR A
07DE 39             RTS
*
07DF 0002           CHPTR RMB 2      PARM LIST
07E1 01EA R         FDB  CHRTAB-520

* TABLE MANIPULATION ROUTINES FOR TABLES
* SYMTAB AND WNTAB
*
* STORAGE LOCATIONS USED BY THE ROUTINES:
*
07E3 0002           PSING1 RMB 2     ADDRESS OF MNEMONIC
07E5 0002           PSING2 RMB 2     ADDRESS IN THE TABLE
07E7 0001           PCOUNT RMB 1     LENGTH OF MNEMONIC
07E6 0702           TABDD RMB 2      TABLE POINTER
07EA 0006           HSMBL RMB 6      SYMBOL TEMP LOC
07F0 0702           HKEYA RMB 2      HASHED CODE
07F2 0702           HKEYB RMB 2      TEMP LOC FOR HASHED CODE
07F4 0702           HSAV1 RMB 2      TEMP LOC FOR PTR
07F6 0002           HSAV2 RMB 2      TEMP LOC FOR PTR
*
*
* STORE A SYMBOL IN SYMTAB
* ON ENTRY DESCRA CONTAINS ADDRESS OF
* THE SYMBOL, AND DESCRC CONTAINS THE LENGTH
* A STANDARD HASH CODED METHOD IS USED
*
*
07Fb BD 08B9 R      STOSYM JSR  HASH    GET HASHED KEY
07Fb FF 02d2 R      STX  SYMPTR      SAVE
*
* SEE IF LOC(HKEYA) IS EMPTY
*
07FE A6 00          SYMA  LDA A 0,X      GET FIRST CHAR
0800 81 20          CMP A #520      BLANK ?
0802 26 2F          BNE  SYMB        NO
*
* STORE SYMBOL IN SYMTAB
*
0804 FF 07F6 R      STX  HSAV2      SAVE TABLE PTR
0807 CE 07EA R      LDX  #HSMBL     POINT TO HSMBL
080A FF 07F4 R      STX  HSAV1      SAVE
080D C6 06          LDA B #6        LOAD SYMBOL LENGTH
*
* DO TRANSFER
*
080F FE 07F4 R      SYM1  LDX  HSAV1    POINT TO HSMBL
0812 A6 00          LDA A 0,X      GET CHAR
0814 08             INX
0815 FF 07F4 R      STX  HSAV1      POINT TO NEXT CHAR
0818 FE 07F6 R      LDX  HSAV2      POINT TO TABLE ENTRY
081B A7 00          STA A 0,X      STORE CHAR IN SYMTAB
081D 08             INX
081E FF 07F6 R      STX  HSAV2      POINT TO NEXT POSITION
0821 5A             DEC B           ALL DONE ?
0822 26 EB          BNE  SYM1        NO
*
* STORE LC, AND SET INFO BYTE
*
0824 B6 0273 R      LDA A LC        GET LC

```

```

0827 A7 00          STA A 0,X      STORE
0829 B6 0274 R      LDA A LC+1     GET LS BYTE OF LC
082C A7 01          STA A 1,X      STORE
082E B6 40          LDA A #540     INFO BYTE:=RELOC,DEFINED
0830 A7 02          STA A 2,X
0832 39             RTS            RETURN

```

```

*
* COMPARE HSMBL WITH ENTRY IN SYMTAB
*
0833 BD 087E R SYMB JSR SYMCMP     COMPARE
0836 26 10          BNE SYMC       NO MATCH
*
* ERROR, SYMBOL ALREADY IN TABLE
*
0838 FE 0282 R      LDX SYMPTR     GET ADDRESS OF ENTRY
083B B6 80          LDA A #580
083D AA 06          ORA A 8,X      SET REDEFINED BIT
083F A7 08          STA A 8,X
0841 CE 0206        LDX #50206     LOAD ERROR#
0844 BD 00BB R SYMB1 JSR PRINTE    PRINT IT
0847 39             RTS            RETURN
*
* FIND ANOTHER SLOT IN SYMTAB FOR SYMBOL
*
0848 BD 0893 R SYMC JSR SYMMOD     GET A(NEXT SLOT)
084B BC 07F0 R      CPX HKEYA      CHECKED ALREADY ?
084E 27 02          BEQ **4        YES, TABLE IS FULL
*
0850 20 AC          BRA SYMA       TRY AGAIN
*
0852 CE 0221        LDX #50221     LOAD ERROR#
0855 20 ED          BRA SYMB1      PRINT IT & RETURN

```

```

* LOOK UP SYMBOL IN SYMTAB
* ON ENTRY DESCRA CONTAINS ADDRESS OF SYMBOL
* AND DESCRC CONTAINS THE LENGTH OF THE
* SYMBOL.
* ON RETURN:
* B=VALUE OF INFO BYTE
* B=FF SYMBOL NOT FOUND
* X=VALUE OF SYMBOL
*

```

```

0857 BD 0889 R LKPSYM JSR HASH     GET KEY
085A FF 0282 R      STX SYMPTR     SAVE

```

```

*
085D A6 00          LKPSM1 LDA A 0,X
085F 81 20          CMP A #520
0861 26 03          BNE LKPSM3     BLANK?
*                                     NO

```

```

* ENTRY NOT IN SYMTAB
*
0863 C6 FF          LKPSM2 LDA B #5FF
0865 39             RTS            LOAD RC
*                                     RETURN

```

```

* COMPARE SYMBOL WITH ENTRY IN SYMTAB
*
0866 BD 087E R LKPSM3 JSR SYMCMP   COMPARE
0869 26 08          BNE LKPSM4     NO MATCH

```

```

* FOUND, EXTRACT INFO, AND VALUE
*
086B FE 0282 R      LDX SYMPTR     POINT TO ENTRY
086E E6 08          LDA B 8,X      GET INFO BYTE
0870 EE 06          LDX 6,X        GET VALUE
0872 39             RTS

```

```

* PROBE AGAIN FOR SYMBOL IN SYMTAB
*
0873 BD 0893 R LKPSM4 JSR SYMMOD   GET NEXT KEY
0876 BC 07F0 R      CPX HKEYA      ALREADY CHECKED?
0879 26 E2          BNE LKPSM1     NO, TRY AGAIN
087B C6 FF          LDA B #5FF     SET RC
087D 39             RTS

```

```

* ROUTINE TO COMPARE SYMBOL WITH ENTRY
*
087E FF 07E3 R SYMCMP STX PSING1   SAVE PTR TO ENTRY
0881 B6 06          LDA A #6
0883 B7 07E7 R      STA A PCOUNT   PCOUNT:=L(SYMBOL)
0886 CE 07EA R      LDX #HSMBL
0889 FF 07E5 R      STX PSING2     POINT TO HSMBL
088C CE 07E3 R      LDX #PSING1    POINT TO PARAMS
088F BD 08C5 R      JSR COMPAR     COMPARE
0892 39             RTS

```

```

*
* FIND NEXT SLOT IN SYMTAB
* SYMPTR:=SYMPTR+9 (MODULO NSYM)
*
0893 FE 0282 R SYMMOD LDX SYMPTR   GET A(CURRENT SLOT)
0896 08             INX            SYMPTR:=SYMPTR+9

```

```

0897 08          INX
0898 08          INX
0899 08          INX
089A 08          INX
089B 08          INX
089C 08          INX
089D 08          INX
089E 08          INX
      *
      * BEYOND SYMTAB ?
      *
089F BC 026C R    CPX    SYMEND
08A2 26 03        BNE    **5      NO
08A4 FE 0268 R    LDX    SYMTAB    POINT TO FIRST ENTRY
08A7 FF 0282 R    STX    SYMPTR    SAVE PTR TO ENTRY
08AA 39          RTS
      *
      * DELETE LAST SYMBOL ENTERED
      *
08AB FE 0282 R    DELSYM LDX    SYMPTR
08AE 46 20        LDA A #520      LOAD BLANK
08B0 C6 09        LDA B #9        LOAD ENTRY LENGTH
      *
08B2 A7 00        DELI   STA A 0,X  BLANK BYTE
08B4 08          INX              POINT TO NEXT BYTE
08B5 5A          DEC B              ALL DONE ?
08B6 26 FA        BNE    DELI      NO
      *
08B8 39          RTS              YES, RETURN
      *
      * HASH SYMBOL TO PRODUCE A KEY
      *
08B9 CE 2020      HASH    LDX    #52020  BLANK HSMBL
08BC FF 07EA R    STX    HSMBL
08BF FF 07EC R    STX    HSMBL+2
08C2 FF 07EE R    STX    HSMBL+4
      *
      * MOVE SYMBOL TO HSMBL
      *
08C5 CE 07EA R    LDX    #HSMBL    POINT TO HSMBL
08C8 FF 07F6 R    STX    HSAV2      SAVE
08CB FE 027B R    LDX    DESCRA    POINT TO SYMBOL
08CE FF 07F4 R    STX    HSAV1      SAVE
08D1 F6 027D R    LDA B DESCRC     GET L (SYMBOL)
      *
08D4 FE 07F4 R    HASHI   LDX    HSAV1  POINT TO SYMBOL
08D7 A6 00        LDA A 0,X        GET CHAR
08D9 08          INX
08DA FF 07F4 R    STX    HSAV1      POINT TO NEXT CHAR
08DD FE 07F6 R    LDX    HSAV2      POINT TO HSYMBL
08E0 A7 00        STA A 0,X        STORE CHAR
08E2 08          INX
08E3 FF 07F6 R    STX    HSAV2      POINT TO NEXT CHAR
08E6 5A          DEC B              ALL DONE?
08E7 26 EB        BNE    HASHI      NO
      *
      * FOLD OVER HSMBL CREATING KEYA
      *
08E9 FE 07EA R    LDX    HSMBL      HKEYA:=HSMBL(2)
08EC FF 07F0 R    STX    HKEYA
08EF FE 07EC R    LDX    HSMBL+2
08F2 FF 07F2 R    STX    HKEYB
08F5 CE 07F0 R    LDX    #HKEYA
08F8 BD 08EC R    JSR    ADD16      +HSMBL+2(2)
08FB FE 07EE R    LDX    HSMBL+4
08FE FF 07F2 R    STX    HKEYB
0901 CE 07F0 R    LDX    #HKEYA
0904 BD 08EC R    JSR    ADD16      +HSMBL+4 (2)
      *
      * HKEYA:=REMAINDER OF HKEYA/NSYM
      *
0907 B6 07F0 R    LDA A HKEYA      LOAD VALUES
090A F6 07F1 R    LDA B HKEYA+1
090D FE 026A R    LDX    NSYM
0910 FF 07F2 R    STX    HKEYB
0913 CE 07F2 R    LDX    #HKEYB     POINT TO NSYM
0916 BD 08A1 R    JSR    DIV16
0919 FF 07F0 R    STX    HKEYA      SAVE REMAINDER
      *
      * HKEYA:=HKEYA*9
      *
091C 4F          CLR A
091D C6 09        LDA B #9
091F CE 07F0 R    LDX    #HKEYA
0922 BD 087D R    JSR    MPY16
0925 B7 07F0 R    STA A HKEYA
0928 F7 07F1 R    STA B HKEYA+1
      *
      * ADD IN BASE ADDRESS OF SYMTAB
      *
092B FE 0268 R    LDX    SYMTAB
092E FF 07F2 R    STX    HKEYB
0931 CE 07F0 R    LDX    #HKEYA

```

```

0934 BD 0BEC R      JSR  ADD16
0937 FE 07F0 R      LDX  HKEYA
093A 39             RTS

* LOOK UP MNEMONIC IN MNTAB
* ON ENTRY DESCRA POINTS TO MNEMONIC, AND
* DESCRC CONTAINS THE LENGTH (3)
* ON RETURN:
* REG A = 00 FOUND
* REG A = FF NOT IN TABLE
* REG X = ADDRESS OF ROUTINE TO PROCESS
*          THE OPCODE/PSEUDOP
* REG B = MACHINE CODE FOR OPCODES
*          = FF FOR PSEUDOPS
*
*
* THE ALGORITHM IS A BINARY SEARCH
*
* TEMPORARY LOCATIONS:
*
093B 0001          LP      RMB 1    ONE BELOW LOWEST ENTRY
093C 0001          MP      RMB 1    ONE HIGHER THAN HIGHEST ENTRY
093D 0001          IP      RMB 1    CALCULATED PROBE VALUE
093E 0006          ENSIZ  FDB 6    LENGTH OF ENTRY IN MNTAB
*
0940 B6 027D R     MNLKP  LDA A DESCRC
0943 B7 07E7 R      STA A PCOUNT    INIT PCOUNT
0946 86 57          LDA A #CHRTAB-MNTAB/6+1  (# ENTRIES+1)
0948 B7 093C R      STA A MP        INIT MP
094B 4F            CLR A
094C B7 093B R      STA A LP        INIT LP
*
094F B6 093B R     MNLKPA LDA A LP
0952 4C            INC A            A:=LP+1
0953 B1 093C R      CMP A MP        MP=LP+1 ?
0956 26 03          BNE  MNLKPB     NO
*
0958 86 FF          LDA A #$FF      YES, ENTRY NOT IN TABLE
095A 39            RTS
*
* IP:=(LP+MP)/2 TRUNCATED
*
095B F6 093B R     MNLKPB LDA B LP
095E FB 093C R      ADD B MP        B:=LP+MP
0961 56            ROR B            B:=B/2
0962 F7 093D R      STA B IP        SAVE IP
*
* GET 16 BIT ADDRESS OF ENTRY
*
0965 4F            CLR A
0966 CE 093E R      LDX  #ENSIZ     GET ENTRY LENGTH
0969 5A            DEC B            B:=IP-1
096A BD 0B7D R      JSR  MPY16      GET (IP-1)*6
096D B7 07E3 R      STA A PSING1    SAVE
0970 F7 07E4 R      STA B PSING1+1
0973 CE 0006 R      LDX  #MNTAB
0976 FF 07E5 R      STX  PSING2     PSING2:=BASE OF MNTAB
0979 CE 07E3 R      LDX  #PSING1    POINT TO PARMS
097C BD 0BEC R      JSR  ADD16      PSING1:=(IP-1)*6+MNTAB
097F FE 07E3 R      LDX  PSING1
0982 FF 07E8 R      STX  TBADD      SAVE
*
* COMPARE MNEMONIC WITH ENTRY IN MNTAB
*
0985 FE 027B R      LDX  DESCRA     GET MNEMONIC ADDRESS
0988 FF 07E5 R      STX  PSING2     INIT PARM FOR COMPARE
098B CE 07E3 R      LDX  #PSING1    POINT TO PARMS
098E BD 06C5 R      JSR  COMPAR     COMPARE
0991 25 0B          BCS  MNLI       ENTRY<MNEMONIC
0993 26 11          BNE  MNMI       ENTRY>MNEMONIC
*
0995 4F            CLR A            ENTRY FOUND
0996 FE 07E8 R      LDX  TBADD      POINT TO ENTRY
0999 E6 05          LDA B 5,X       GET MC
099B EE 03          LDX  3,X        GET BRANCH ADDRESS
099D 39            RTS
*
* ENTRY<MNEMONIC LP:=IP
*
099E B6 093D R     MNLI  LDA A IP
09A1 B7 093B R      STA A LP
09A4 20 A9          BRA  MNLKPA     TRY AGAIN
*
* ENTRY>MNEMONIC MP:=IP
*
09A6 B6 093D R     MNMI  LDA A IP
09A9 B7 093C R      STA A MP
09AC 20 A1          BRA  MNLKPA     TRY AGAIN
*
* EVALUATE NUMBERS, SYMBOLS AND EXPRESSIONS
*
*
09AE 0002          VALUE  RMB 2      TEMPORARY LOCS

```

09B0	0002		IMPVAL	RMB	2	
09B2	0001		CLFLG	RMB	1	CLASS OF PREVIOUS TOKEN
09B3	0001		CLASS	RMB	1	CLASS OF CURRENT TOKEN
09B4	0001		OPERN	RMB	1	ARITHMETIC OPERATOR
*						
09B5	7F 09AE	R	NSEVL	CLR	VALUE	
09B8	7F 09AF	R		CLR	VALUE+1	VALUE:=0
09BB	7F 09B2	R		CLR	CLFLG	CLFLG:=0
09BE	B7 09B3	R		STA	A CLASS	SAVE CLASS OF CURRENT TOKEN
09C1	C1 2A			CMP	B #52A	* ?
09C3	26 2U			BNE	NSVLCI	NO
*						
09C5	FE 0273	R		LDX	LC	YES
09C8	FF 09AE	R		STX	VALUE	VALUE:=LC
09CB	86 02			LDA	A #2	
09CD	B7 09B2	R		STA	A CLFLG	CLFLG:=2
09D0	73 0277	R		COM	REFLG	REFLG:=RELOC
*						
09D3	FE 027E	R	NSVLA	LDX	CUCHAR	
09D6	A6 00			LDA	A 0,X	GET NEXT CHAR
09D8	81 20			CMP	A #520	BLANK?
09DA	27 08			BEQ	NSVLB	YES
09DC	81 00			CMP	A #50D	EOL?
09DE	27 04			BEQ	NSVLB	YES
09E0	81 2C			CMP	A #52C	" , " ?
09E2	26 08			BNE	NSVLC	NO
*						
09E4	FE 09AE	R	NSVLB	LDX	VALUE	
09E7	FF 0C68	R		STX	ADRI	ADRI,2:=VALUE
09EA	5F			CLR	B	RC:=00
09EB	39			RTS		ALL DONE
*						
09EC	BD 06F6	R	NSVLC	JSR	NXTOK	GET NEXT TOKEN
09EF	B7 09B3	R		STA	A CLASS	SAVE CLASS
09F2	B1 09B2	R	NSVLCI	CMP	A CLFLG	CLASS=CLFLG?
09F5	26 06			BNE	NSVLF	NO
*						
09F7	CE 0204		NSVLD	LDX	#50204	ERROR
09FA	5F		NSVLE	CLR	B	
09FB	53			COM	B	RC:=FF
09FC	39			RTS		RETURN
*						
09FD	81 02		NSVLF	CMP	A #502	STRING?
09FF	27 14			BEQ	NSVLH	YES
*						
0A01	81 24			CMP	A #524	ARITHMETIC OPERATOR?
0A03	27 02			BEQ	NSVLG	YES
*						
0A05	20 F0			BRA	NSVLD	ERROR
*						
0A07	7D 09B2	R	NSVLG	IST	CLFLG	CLFLG=0?
0A0A	27 EB			BEQ	NSVLD	YES,ERROR
*						
0A0C	F7 09B4	R		STA	B OPERN	SAVE OPERATOR
0A0F	B7 09B2	R		STA	A CLFLG	CLFLG:=CLASS
0A12	7E 09D3	R		JMP	NSVLA	SCAN AGAIN
*						
0A15	C1 03		NSVLH	CMP	B #503	HEX STRING?
0A17	26 11			BNE	NSVLJ	NO
*						
0A19	F6 027D	R		LDA	B DESCRC	YES
0A1C	C1 04			CMP	B #4	> 4 ?
0A1E	2F 05			BLE	NSVLHI	NO
*						
0A20	CE 0210			LDX	#50210	YES,ERROR
0A23	20 05			BRA	NSVLE	
*						
0A25	BD 0ACE	R	NSVLHI	JSR	CVHB	CONVERT
0A28	20 3B			BRA	NSVLM	
*						
0A2A	C1 09		NSVLJ	CMP	B #9	DECIMAL?
0A2C	26 11			BNE	NSVLK	NO
*						
0A2E	F6 027D	R		LDA	B DESCRC	
0A31	C1 05			CMP	B #5	> 5 ?
0A33	2F 05			BLE	NSVLJI	NO
*						
0A35	CE 0210			LDX	#50210	YES,ERROR
0A38	20 C0			BRA	NSVLE	
*						
0A3A	BD 0B2A	R	NSVLJI	JSR	CVDB	CONVERT
0A3D	20 26			BRA	NSVLM	
*						
0A3F	C1 01		NSVLK	CMP	B #501	SYMBOL?
0A41	27 03			BEQ	NSVLL	YES
*						
0A43	7E 09F7	R		JMP	NSVLD	NO,ERROR
*						
0A46	BD 0B57	R	NSVLL	JSR	LKPSYM	L(X)KUP SYMBOL
0A49	C5 80			BIT	B #580	REDEFINED ?
0A4B	26 12			BNE	NSVLLA	YES

0A4D C5 40	*	BIT B #540	RELOC ?
0A4F 27 05		BEQ **7	NO
0A51 73 0277 R	*	COM RELFLG	YES RELFLG*=RELOC
0A54 20 0F		BRA NSVLM	
0A56 C5 10	*	BIT B #510	COMMON?
0A58 27 03		BEQ **5	NO
0A5A 73 0278 R	*	COM CMNFLG	YES
0A5D 20 06		BRA NSVLM	
0A5F CE 0211	*	NSVLLA LDX #0211	NO, ERROR
0A62 7E 09FA R		JMP NSVLE	
0A65 FF 09B0 R	*	NSVLM STX TMPVAL	SAVE, CONVERTED VALUE
0A68 7D 09B2 R		IST CLFLG	CLFLG=0 ?
0A6B 26 0F		BNE NSVLP	NO
0A6D FE 09B0 R	*	LDX TMPVAL	YES
0A70 FF 09AE R		STX VALUE	VALUE*=TMPVAL
0A73 B6 09B3 R	*	NSVLN LDA A CLASS	
0A76 B7 09B2 R		STA A CLFLG	CLFLG*=CLASS
0A79 7E 09D3 R		JMP NSVLA	SCAN AGAIN
0A7C B6 09B4 R	*	NSVLP LDA A OPERN	GET LAST OPERATOR
0A7F 81 2B		CMP A #52B	+ ?
0A81 26 08		BNE NSVLP1	NO
0A83 CE 09AE R	*	LDX #VALUE	
0A86 BD 0BEC R		JSR ADD16	VALUE*=VALUE+TMPVAL
0A89 20 E8		BRA NSVLN	
0A8B 81 2D	*	NSVLP1 CMP A #52D	- ?
0A8D 26 08		BNE NSVLP2	NO
0A8F CE 09AE R	*	LDX #VALUE	YES
0A92 BD 0BFD R		JSR SUB16	VALUE*=VALUE-TMPVAL
0A95 20 DC		BRA NSVLN	
0A97 81 2A	*	NSVLP2 CMP A #52A	* ?
0A99 26 15		BNE NSVLP3	NO
0A9B B6 09AE R	*	LDX #VALUE	
0A9E F6 09AF R		LDA B VALUE+1	
0AA1 CE 09B0 R		LDX #TMPVAL	
0AA4 BD 0B7D R		JSR MPY16	VALUE*=VALUE*TMPVAL
0AA7 B7 09AE R		STA A VALUE	
0AAA F7 09AF R		STA B VALUE+1	
0AAD 7E 0A73 R		JMP NSVLN	
0AB0 81 2F	*	NSVLP3 CMP A #52F	/ ?
0AB2 27 03		BEQ NSVLP4	YES
0AB4 7E 09F7 R	*	JMP NSVLD	NO, ERROR
0AB7 B6 09AE R	*	NSVLP4 LDA A VALUE	
0ABA F6 09AF R		LDA B VALUE+1	
0ABD CE 09B0 R		LDX #TMPVAL	
0ACO BD 0BA1 R		JSR DIV16	VALUE*=VALUE/TMPVAL
0AC3 B7 09AE R		STA A VALUE	
0AC6 F7 09AF R		STA B VALUE+1	
0AC9 7E 0A73 R		JMP NSVLN	
	*	CVHB CONVERT HEX TO BINARY	
	*	ON ENTRY DESCRA = ADDRESS OF STRING	
	*	DESCRC = # OF BYTES IN STRING	
	*	ON RETURN [X]=VALUE	
0ACC 0002	*	HVAL RMB 2	TEMP STORAGE
0ACE FE 027B R	*	CVHB LDX DESCRA	GET ADDRESS OF STRING
0AD1 7F 0ACC R		CLR HVAL	
0AD4 7F 0ACD R		CLR HVAL+1	
0AD7 F6 027D R		LDA B DESCRC	GET COUNT
0ADA 09		DEX	DECREMENT PTR TO STRING
0ADB 08	CVHB1	INX	POINT TO RIGHT MOST
0ADC 5A		DEC B	BYTE OF THE
0ADD 26 FC		BNE CVHB1	STRING
0AD7 F6 027D R	*	LDA B DESCRC	GET COUNT
0AE2 BD 0B1B R		JSR CVHBS	CONVERT
0AE5 B7 0ACD R		STA A HVAL+1	SAVE
0AE8 5A		DEC B	DECREMENT COUNT
0AE9 27 29		BEQ CVHBD	(1 HEX DIGIT)
0AEB 09		DEX	POINT TO NEXT LEFT BYTE
0AEC BD 0B1B R		JSR CVHBS	CONVERT
0AEF 48		ASL A	SHIFT TO LEFT NIBBLE
0AF0 48		ASL A	
0AF1 48		ASL A	


```

0AF2 4B      ASL A
0AF3 BA OACD R  ORA A HVAL+1  CONVERT TO BYTE
0AF6 B7 OACD R  STA A HVAL+1  SAVE
0AF9 5A      DEC B      DECREMENT COUNT
0AFA 27 18     BEQ   CVHBD   (2 HEX DIGITS)
0AFC 09      DEX          POINT TO NEXT LEFT BYTE
0AFD BD OB18 R JSR   CVHBS   CONVERT
0B00 B7 OACC R STA A HVAL  SAVE
0B03 5A      DEC B      DECREMENT COUNT
0B04 27 0E     BEQ   CVHBD   (3 HEX DIGITS)

*
0B06 09      DEX          POINT TO NEXT LEFT BYTE
0B07 BD OB18 R JSR   CVHBS   CONVERT
0B0A 4B      ASL A      SHIFT TO LEFT NIBBLE
0B0B 4B      ASL A
0B0C 4B      ASL A
0B0D 4B      ASL A
0B0E BA OACC R  ORA A HVAL  CONVERT TO BYTE
0B11 B7 OACC R  STA A HVAL  SAVE
0B14 FE OACC R CVHBD LDX HVAL GET FINAL VALUE
0B17 39      RTS          RETURN

*
* ROUTINE TO CONVERT ASCII TO BINARY
*
0B18 A6 00     CVHBS LDA A 0,X  GET BYTE
0B1A 80 30     SUB A #50      CONVERT
0B1C 81 09     CMP A #509     0 - 9 ?
0B1E 2F 02     BLE **4       YES
0B20 83 07     SUB A #507     NO, 10 - 15
0B22 39      RTS

*
* CVDB: CONVERT DECIMAL TO BINARY
* ON ENTRY DESCRA = ADDRESS OF DECIMAL STRING
* DESCRC = # BYTES IN DECIMAL STRING
* ON RETURN (X) = VALUE IN BINARY
*
0B23 0002     DVAL RMB 2      TEMP STORAGE FOR BINARY
0B25 0001     DCOUNT RMB 1     DIGIT COUNT
0B26 0002     TENVL RMB 2      POWER OF TEN
0B28 0002     DXSAV RMB 2      TEMPORARY STORAGE FOR X
*
0B2A 7F OB23 R CVDB CLR DVAL  DVAL:=0
0B2D 7F OB24 R CLR DVAL+1
0B30 7F OB26 R CLR TENVL
0B33 7F OB27 R CLR TENVL+1
0B36 7C OB27 R INC TENVL+1  TENVL:=1
0B39 FE 027B R LDX DESCRA  POINT TO STRING
0B3C 09      DEX
0B3D F6 027D R LDA B DESCRC
0B40 F7 0B25 R STA B DCOUNT  INIT DCOUNT

*
0B43 08      CVDB1 INX       POINT TO
0B44 5A      DEC B      LEAST SIGNIFICANT
0B45 26 FC     BNE CVDB1  DIGIT

*
0B47 FF OB28 R CVDB2 STX DXSAV  SAVE POINTER
0B4A E6 00     LDA B 0,X  GET DIGIT
0B4C C4 0F     AND B #50F  CONVERT TO BCD
0B4E 4F      CLR A      CLEAR ACCUMULATOR
0B4F CE OB26 R LDX #TENVL  POINT TO POWER OF TEN
0B52 BD OB7D R JSR MPY16   (A,B):=TENVL*DIGIT
0B55 FB OB24 R ADD B DVAL+1  DVAL:=DVAL+TENVL*DIGIT
0B58 B9 OB23 R ADC A DVAL
0B5B B7 OB23 R STA A DVAL
0B5E F7 OB24 R STA B DVAL+1
0B61 4F      CLR A
0B62 C6 0A     LDA B #50A   B:=10
0B64 CE OB26 R LDX #TENVL  POINT TO POWER OF TEN
0B67 BD OB7D R JSR MPY16   TENVL:=TENVL*10
0B6A B7 OB26 R STA A TENVL
0B6D F7 OB27 R STA B TENVL+1
0B70 FE OB28 R LDX DXSAV  RESTORE POINTER TO STRING
0B73 09      DEX          POINT NEXT LEFT DIGIT
0B74 7A OB25 R DEC DCOUNT  DONE?
0B77 26 CE     BNE CVDB2   NO
0B79 FE OB23 R LDX DVAL    GET FINAL VALUE
0B7C 39      RTS          RETURN

*
* MPY16 16 BIT MULTIPLY ROUTINE
* (A,B):=(A,B)*(2 BYTES POINTED AT BY X REG)
* USES 7 BYTES ON THE STACK
*
0B7D 37      MPY16 PSH B      PUT VALUES ON TO THE STACK
0B7E 36      PSH A
0B7F A6 01     LDA A 1,X
0B81 36      PSH A
0B82 A6 00     LDA A 0,X
0B84 36      PSH A
0B85 86 10     LDA A #16
0B87 36      PSH A
0B88 30      TSX          POINT TO DATA
0B89 A6 03     LDA A 3,X
*

```

```

OB8B 58      MPY163 ASL B
OB8C 49      ROL A          FORM ANSWER
OB8D 68 02   ASL 2,X        SHIFT MULTIPLICAND
OB8F 69 01   ROL 1,X
OB91 24 04   BCC MPY167
OB93 EB 04   ADD B 4,X      ADD MULTIPLIER
OB95 A9 03   ADC A 3,X
OB97 6A 00   MPY167 DEC 0,X
OB99 26 F0   BNE MPY163    COUNT NOT ZERO
OB9B 31      INS
OB9C 31      INS
OB9D 31      INS
OB9E 31      INS
OB9F 31      INS
OBA0 39      RTS          ALL DONE

* DIV16 16 BIT DIVIDE (UNSIGNED)
* (A,B)*=(A,B)/(X),(X+1)
* [X]=REMAINDER
*
OBA1 37      DIV16 PSH B      DIVIDEND TO STACK
OBA2 36      PSH A
OBA3 A6 00   LDA A 0,X
OBA5 E6 01   LDA B 1,X
OBA7 37      PSH B      DIVISOR TO STACK
OBA8 36      PSH A
OBA9 34      DES          LEAVE ROOM FOR COUNT
OBAA 30      ISX          (X) PTR TO STACKED DATA
OBAE 86 01   LDA A #1
OBAF 6D 01   IST 1,X
OBAF 2B 0B   BMI DIV153
OBB1 4C      DIV151 INC A
OBB2 68 02   ASL 2,X
OBB4 69 01   ROL 1,X
OBB6 2B 04   BMI DIV153
OBB8 81 11   CMP A #17
OBBA 26 F5   BNE DIV151
OBBE A7 00   DIV153 STA A 0,X    SAVE COUNT
OBBE A6 03   LDA A 3,X
OBC0 E6 04   LDA B 4,X
OBC2 6F 03   CLR 3,X
OBC4 6F 04   CLR 4,X
OBC6 E0 02   DIV163 SUB B 2,X
OBC8 A2 01   SBC A 1,X
OBCA 24 07   BCC DIV165    DIVISOR STILL OK
OBCB EB 02   ADD B 2,X      DIVISOR TOO LARGE
OBCD A9 01   ADC A 1,X      RESTORE
OBDE 0C      CLC
OBD1 20 01   BRA DIV167
OBD3 0D      DIV165 SEC
OBD4 69 04   DIV167 ROL 4,X
OBD6 69 03   ROL 3,X
OBD8 64 01   LSR 1,X      ADJUST DIVISOR
OBD8 64 01   LSR 1,X
OBD8 64 01   ROR 2,X
OBD8 64 01   DEC 0,X
OBD8 64 01   BNE DIV163
OBE0 A7 00   * STA A 0,X    SAVE REMAINDER IN X
OBE2 E7 01   STA B 1,X
OBE4 EE 00   LDX 0,X
OBE6 31      INS          CLEAN UP STACK
OBE7 31      INS
OBE8 31      INS
OBE9 32      PUL A
OBEA 33      PUL B
OBEB 39      RTS

* ADD16 16 BIT ADDITION
* [X] POINTS:
* LOC(2),TEMP(2)
* LOC(2)*=LOC(2)+TEMP(2)
*
OBE C 36     ADD16 PSH A
OBE D 37     PSH B
OBE E A6 01  LDA A 1,X
OBE F E6 00  LDA B 0,X
OBF 2 A8 03  ADD A 3,X
OBF 4 E9 02  ADC B 2,X
OBF 6 A7 01  STA A 1,X
OBF 8 E7 00  STA B 0,X
OBF A 33     PUL B
OBF B 32     PUL A
OBF C 39     RTS

*
*
* SUB16 16 BIT SUBTRACTION
* [X] POINTS:
* LOC(2),TEMP(2)
* LOC(2)*=LOC(2)-TEMP(2)
*
OBF D 36     SUB16 PSH A
OBF E 37     PSH B
OBF F A6 01  LDA A 1,X
OC0 1 E6 00  LDA B 0,X

```

```

OC03 A0 03          SUB A 3,X
OC05 E2 02          SBC B 2,X
OC07 A7 01          SIA A 1,X
OC09 E7 00          SIA B 0,X
OC0B 33             PUL B
OC0C 32             PUL A
OC0D 39             RTS

* PRINT A LINE ON THE TTY
*
OC0E B6 026E R PRINTL LDA A OPINS      GET OPTIONS
OC11 B5 B0          BIT A #580         LIST?
OC13 26 14          BNE PLEND          NO
OC15 7D 0275 R      ISL PASS          PASS?
OC18 27 0F          BEQ PLEND          PASS?
OC1A 7D 030C R      ISL MACFLG        MACRO FLAG SET?
OC1D 27 04          BEQ PRINTI        NO

*
OC1F B5 10          BIT A #510         PRINT MACROS?
OC21 26 06          BNE PLEND          NO

*
OC23 BD OC2A R PRINTI JSR LINCK        CHECK LINE #
OC26 BD OC71 R      JSR OUTL          PRINT A LINE
OC29 39             PLEND RTS          ALL DONE

*
* LINE CHECK FOR TOP OF PAGE ETC.
*
OC2A 37             LINCK PSH B
OC2B F6 02d7 R      LDA B LCOUNT
OC2E C1 00          CMP B #500         END OF PAGE?
OC30 26 03          BNE LINCKA        NO
OC32 BD OC44 R      JSR SPACER        YES SPACE TO TOP OF PAGE
OC35 7C 02d7 R      LINCKA INC LCOUNT BUMP LCOUNT
OC38 F6 02d7 R      LDA B LCOUNT
OC3B C1 3C          CMP B #53C        LCOUNT=60?
OC3D 26 03          BNE LINCKB        NO
OC3F 7F 02d7 R      CLR LCOUNT      YES, SET FOR TOP OF PAGE
OC42 33             LINCKB PUL B
OC43 39             RTS

*
* SPACE TO TOP OF PAGE AND PRINT PAGE MARK
*
OC44 CE OC4B R SPACER LDX #HEADR      POINT TO DATA
OC47 BD 1861 R      JSR PDATAI       PRINT ON TTY
OC4A 39             RTS

*
OC4B 0D0A          HEADR FDB $0D0A CRLF
OC4D 0D0A          FDB $0D0A
OC4F 0D0A          FDB $0D0A
OC51 2E            FCC '.....'
OC5E 0D0A          FDB $0D0A
OC60 0D0A          FDB $0D0A
OC62 0D0A          FDB $0D0A
OC64 04            FCB $04 EOT

* PRINT A FORMATTED LINE OF LISTING ON THE TTY
*
OC65 0701          MCOUNT RMB 1      # BYTES OF MACHINE CODE
OC66 0001          POP RMB 1          PSEUDOP10=NO:1,2 BYTES
OC67 0001          OPCD RMB 1         OPCODE IN HEX
OC68 0701          ADRI RMB 1         INSTRUCTION ADDRESS
OC69 0701          ADRI RMB 1
OC6A 0005          LINEN RMB 5        LINENUM IN ASCII
OC6C 2004          FDB $2004          EOT

*
OC71 CE OC6A R OUTL LDX #LINEN        LOAD PARMS
OC74 B6 026F R      LDA A LNUM        LOAD LINNUM (BINARY)
OC77 F6 0270 R      LDA B LNUM+1
OC7A BD 0D5C R      JSR CVBTD        CONVERT TO DECIMAL (ASCII)
OC7D CE OC6B R      LDX #LINEN+1     POINT TO DECIMAL LINE#
OC80 BD 1861 R      JSR PDATAI       PRINT LINE#

*
OC83 7D 030C R      ISL MACFLG        MACRO LINE?
OC86 27 05          BEQ **7          NO

*
OC8B B6 2B          LDA A #7+
OC8A BD 1888 R      JSR OUTCHR

*
OC8D CE 0D59 R      LDX #BLANK6
OC90 BD 1861 R      JSR PDATAI       PRINT 2 BLANKS

*
OC93 7D OC65 R      ISL MCOUNT      PRINT LC ?
OC96 26 00          BNE OUTLA        YES
OC98 7D OC66 R      ISL POP          PRINT LC?
OC9B 26 08          BNE OUTLA        YES

*
OC9D CE 0D56 R      LDX #BLANK3      NO, PRINT BLANKS (5)
OCA0 BD 1861 R      JSR PDATAI
OCA3 20 2B          BRA OUTL2

*
OCA5 CE 0273 R OUTLA LDX #LC          POINT TO LC
OCA8 BD 1871 R      JSR OUT4HS       PRINT IN HEX, SPACE

```

OCAB F6 0C66 R	LDA B POP	PSEUDOP?
OCAE 27 20	BEQ OUTL2	NO
OCB0 C1 01	CMP B #S01	1 BYTE?
OCB2 27 0E	BEQ OUTL1	YES
OCB4 CE 0C68 R	LDX #ADR1	POINT TO ADR1,ADR2
OCB7 BD 1871 R	JSR OUT4HS	PRINT 2 BYTES 4 HEX,SPACE
OCBA CE 0D58 R	LDX #BLANK5	POINT TO BLANKS
OCBD BD 1861 R	JSR PDATA1	PRINT BLANKS
OCC0 20 45	BRA OUTL6	
*		
OCC2 CE 0C69 R OUTL1	LDX #ADR2	POINT TO ADR2
OCC5 BJ 1873 R	JSR OUT2HS	PRINT 1 BYTE 2 HEX,SPACE
OCC8 CE 0D56 R	LDX #BLANK3	POINT TO BLANKS
OCCb BD 1861 R	JSR PDATA1	PRINT BLANKS
OCCe 20 37	BRA OUTL6	
OCU0 F6 0C65 R OUTL2	LDA B MCOUNT	PRINT NOTHING?
OCU3 26 03	BNE OUTL3	NO
OCU5 CE 0D53 R	LDX #BLANK	PRINT JUST 8 BLANKS
OCU8 BD 1861 R	JSR PDATA1	
OCU8 20 2A	BRA OUTL6	
*		
OCU0 CE 0C67 R OUTL3	LDX #OPCD	
OCE0 BD 1873 R	JSR OUT2HS	PRINT OPCODE(HEX),SPACE
OCE3 C1 01	CMP B #S01	ONLY OPCODE?
OCE5 26 06	BNE OUTL4	NO
OCE7 CE 0D56 R	LDX #BLANK3	PRINT BLANKS
OCEA BD 1861 R	JSR PDATA1	
OCEd 20 18	BRA OUTL6	
*		
OCEF C1 02 OUTL4	CMP B #S02	1 BYTE ADDRESS?
OCF1 26 0E	BNE OUTL5	NO,2 BYTES
OCF3 CE 0C69 R	LDX #ADR2	POINT TO ADR2
OCF6 BD 1873 R	JSR OUT2HS	PRINT 1 BYTE ADDRESS,SPACE
OCF9 CE 0D59 R	LDX #BLANK6	PRINT BLANKS
OCFC BD 1861 R	JSR PDATA1	
OCFF 20 06	BRA OUTL6	
*		
OD01 CE 0C68 R OUTL5	LDX #ADR1	POINT TO ADR1,ADR2
OD04 BD 1871 R	JSR OUT4HS	PRINT 2 BYTE ADDRESS,SPACE
*		
OD07 7D 0278 R OUTL6	TST CMNFLG	COMMON?
OD0A 27 04	BEQ **6	NO
*		
OD0C 86 43	LDA A #'C	
OD0E 20 1D	BRA OUTL6B	
*		
OD10 7D 0279 R	TST EXIFLG	EXTERNAL?
OD13 27 04	BEQ **6	NO
*		
OD15 86 58	LDA A #'X	
OD17 20 14	BRA OUTL6B	
*		
OD19 7D 027A R	TST ENTFLG	ENTRY?
OD1C 27 04	BEQ **6	NO
*		
OD1E 86 4E	LDA A #'N	
OD20 20 08	BRA OUTL6B	
*		
OD22 7D 0277 R	TST RELFLG	RELOCATABLE?
OD25 27 04	BEQ **6	NO
*		
OD27 86 52	LDA A #'R	
OD29 20 02	BRA OUTL6B	
*		
*		
OD2B 86 20 OUTL6A	LDA A #S20	LOAD SPACE
OD2D BD 1888 R OUTL6B	JSR OUTCHR	PRINT (A)
OD30 86 20	LDA A #S20	LOAD SPACE
OD32 BD 1888 R	JSR OUTCHR	PRINT SPACE
OD35 FE 0280 R OUTL7A	LDX CULINE	POINT TO LINE
OD38 A6 00 OUTL7	LDA A 0,X	GET CHAR
OD3A 36	PSH A	SAVE A
OD3B BD 1888 R	JSR OUTCHR	PRINT CHAR
OD3E 08	INX	BUMP POINTER
OD3F 32	PUL A	RESTORE A
OD40 81 0D	CMP A #S0D	CR?
OD42 26 F4	BNE OUTL7	NO
OD44 86 0A	LDA A #S0A	YES
OD46 BD 1888 R	JSR OUTCHR	PRINT LF
OD49 7F 0C66 R	CLR POP	
OD4C 7F 0C65 R	CLR MCOUNT	
OD4F 7F 0277 R	CLR RELFLG	
OD52 39	RTS	
*		
OD53 20	BLANK FCB \$20	BLANKS:
OD54 20	FCB \$20	
OD55 20	FCB \$20	
OD56 20	BLANK3 FCB \$20	
OD57 20	FCB \$20	
OD58 20	BLANK5 FCB \$20	
OD59 20	BLANK6 FCB \$20	
OD5A 20	FCB \$20	

```

0D5B 04          FCB  $04          EOT
* CONVERT BINARY 16 BITS TO 5 DECIMAL CHARS
* ON ENTRY (A,B) = 16 BIT BINARY VALUE
* [X] = ADDRESS OF 5 BYTE STRING FOR DECIMAL
* (ASCII) CONVERTED VALUE.
*
0D5C FF 0D9D R CVBITD STX  SAVEX  SAVE DATA PTR
0D5F CE 0D92 R      LUX  #K10K  LOAD PTR TO CONSTANTS
0D62 7F 0D9C R CVDEC1 CLR  SAVEA  INIT DEC CHAR
0D65 E0 01      CVDEC2 SUB  B 1,X
0D67 A2 00      SBC  A 0,X
0D69 25 05      BCS  CVDEC5  OVERFLOW
0D6B 7C 0D9C R  INC  SAVEA  BUMP CHAR BEING BUILT
0D6E 20 F5      BRA  CVDEC2
*
0D70 EB 01      CVDEC5 ADD  B 1,X  RESTORE PARTIAL RESULT
0D72 A9 00      ADC  A 0,X
0D74 36      PSH  A
0D75 FF 0D9F R  STX  SAVEX1
0D76 FE 0D9D R  LDX  SAVEX  LOAD STORE CHAR PTR
0D76 B6 0D9C R  LDA  A SAVEA
0D7E 88 30      ADD  A #S30  MAKE ASCII CHAR
0D80 A7 00      STA  A 0,X
0D82 32      PUL  A
0D83 08      INX
0D84 FF 0D9D R  STX  SAVEX
0D87 FE 0D9F R  LDX  SAVEX1  LOAD PTR TO CONSTANTS
0D8A 08      INX
0D8B 08      INX
0D8C 8C 0D9C R  CPX  #K10K+10
0D8F 26 D1      BNE  CVDEC1
0D91 39      RTS
*
* CONSTANTS
0D92 2710      K10K  FDB  10000
0D94 03E8      FDB  1000
0D96 0064      FDB  100
0D98 000A      FDB  10
0D9A 0001      FDB  1
*
* TEMPORARY STORAGE
*
0D9C 0001      SAVEA  RMB  1
0D9D 0002      SAVEX  RMB  2  STORE DATA PTR
0D9F 0002      SAVEX1 RMB  2  PTR TO CONSTANTS
*
* PRINT ERROR MESSAGES ROUTINE *
* ON ENTRY [X] = ERROR# IN BCD *
*
0DA1 0702      ERNUM  RMB  2  ERROR # IN BCD
0DA3 2A      ERMSA  FCC  '**** ERROR# '
0DAF 0703      ERMSB  RMB  3  ERROR # IN ASCII
0DB2 20      FCB  $20  BLANK
0DB3 0005      ERMSC  RMB  5  ERROR# IN ASCII
0DB6 20      FCB  $20  BLANK
0DB9 3A      FCC  '!'
0DBA 04      FCB  $04  EOT
*
0D3B 36      PRINT  PSH  A
0D3C 37      PSH  B
0D3D FF 0DA1 R  STX  ERNUM  SAVE ERROR #
0DC0 B6 0DA1 R  LDA  A ERNUM  GET ERROR #
0DC3 83 30      ADD  A #S30  CONVERT TO ASCII
0DC5 B7 0DAF R  STA  A ERMSB  SAVE
0DC6 B6 0DA2 R  LDA  A ERNUM+1 GET ERROR #
0DC8 44      LSR  A  SHIFT TO RIGHT NIBBLE
0DCC 44      LSR  A
0DCE 44      LSR  A
0DCF 88 30      AND  A #S30  CONVERT TO ASCII
0DD1 B7 0DB0 R  STA  A ERMSB+1 SAVE
0DD4 B6 0DA2 R  LDA  A ERNUM+1 GET ERROR#
0DD7 84 0F      AND  A #SOF  MASK OUT LEFT NIBBLE
0DD9 88 30      AND  A #S30  CONVERT TO ASCII
0DDB B7 0DB1 R  STA  A ERMSB+2 SAVE
0DDE CE 0DB3 R  LDX  #ERMSC  POINT TO LNUM AREA
0DE1 B6 020F R  LDA  A LNUM
0DE4 F6 0270 R  LJA  B LNUM+1
0DE7 B0 0D5C R  JSR  CVBITD  CONVERT LNUM TO DECIMAL
0DEA CE 0DA3 R  LDX  #ERMSA  PRINT MESSAGE
0DEB B0 1361 R  JSR  PDATA1
0DF0 B0 0D35 R  JSR  OUTL7A  PRINT LAST PART OF LINE
0DF3 33      PUL  B
0DF4 32      PUL  A
0DF5 FE 0208 R  LDX  ECOUNT  BUMP ECOUNT
0DF8 08      INX
0DF9 FF 0288 R  STX  ECOUNT
0DFC FE 0DA1 R  LDX  ERNUM
0DFE 39      RTS
*
**ADDRESS TYPE 1**
*
* [ADC ADD AND BIT CMP EOR LDA ORA SBC SUB]

```

```

*
* IMMEDIATE (2 BYTES):
* CCC A #NUMBER      CCC B #NUMBER
* CCC A #SYMBOL      CCC B #SYMBOL
* CCC A #EXPRESSION  CCC B #EXPRESSION
* CCC A #C           CCC B #C
*
* DIRECT (2 BYTES) OR EXTENDED (3 BYTES):
* CCC A NUMBER      CCC B NUMBER
* CCC A SYMBOL      CCC B SYMBOL
* CCC A EXPRESSION  CCC B EXPRESSION
*
* INDEXED (2 BYTES):
* CCC A NUMBER,X    CCC B NUMBER,X
* CCC A SYMBOL,X    CCC B SYMBOL,X
* CCC A EXPRESSION,X CCC B EXPRESSION,X
*
OE00 B0 1089 R ADDR1 JSR  ADRINT  INIT ADDRESS FIELD VALUES
OE05 B0 06F6 R      JSR  NXTOK   GET NEXT TOKEN
OE06 C1 00      CMP  B #S0D     EOL?
OE08 26 08      BNE  ADDR1B     NO
*
OE0A CE 0204 ADDR1A LDX  #S0204  ERROR
OE0D B0 000B R      JSR  PRINTE  PRINT
OE10 20 5B      BRA  ADDR1E     RETURN
*
OE12 B0 10B7 R ADDR1B JSR  ABRCK  CHECK FOR REGISTER A OR B
OE15 F6 1084 R      LDA  B ABR   NEITHER?
OE16 27 F0      BEQ  ADDR1A     YES ERROR
OE1A B0 06F6 R      JSR  NXTOK   GET NEXT TOKEN
OE1D C1 23      CMP  B #S23     IMMED. MODE?
OE1F 26 14      BNE  ADDR1C     NO
OE21 73 1085 R      COM  IMMED   SET IMMEDIATE FLAG
OE24 B0 06F6 R      JSR  NXTOK   GET NEXT TOKEN
OE27 C1 27      CMP  B #S27     "?? ?
OE29 26 0A      BNE  ADDR1C     NO
*
OE2B FE 027E R      LDX  CUCHAR  GET NEXT CHAR
OE2E A0 00      LDA  A 0,X
OE30 B7 0C69 R      STA  A ADR2
OE33 20 0B      BRA  ADDR1K
*
OE35 B0 09B5 R ADDR1C JSR  NSEVL  EVALUATE OPERAND
OE36 B0 10D7 R      JSR  P2ERR   PRINT PASS 2 ERRORS
OE3B F6 1085 R      LDA  B IMMED  IMMEDIATE MODE?
OE3E 27 0C      BEQ  ADDR1D     NO
OE40 C6 80 ADDR1K LDA  B #S80    IMMEDIATE FORM A
OE42 F7 1087 R      STA  B ORBYA  NIBBLE
OE45 C6 C0      LDA  B #SC0     OF
OE47 F7 1088 R      STA  B ORBYB  MACHINE CODE
OE4A 20 3C      BRA  ADDR1H
*
OE4C B0 06F6 R ADDR1D JSR  NXTOK  GET NEXT TOKEN
OE4F B0 10C4 R      JSR  INXCK   INDEXED?
OE52 26 2A      BNE  ADDR1G     YES
*
OE54 7D 0278 R      1ST  CMNFLG  COMMON?
OE57 26 0A      BNE  ADDR1L     YES
*
OE59 7D 0277 R      1ST  RELFLG  RELOC ?
OE5C 26 05      BNE  ADDR1L     YES
*
OE5E F6 0C68 R      LDA  B ADRI   DIRECT?
OE61 27 0F      BEQ  ADDR1F     YES
*
OE63 C6 B0 ADDR1L LDA  B #SB0    EXTENDED, FORM A
OE65 F7 1087 R      STA  B ORBYA  NIBBLE
OE68 C6 F0      LDA  B #SFO     OF
OE6A F7 1088 R      STA  B ORBYB  MACHINE CODE
*
OE6D B0 115E R ADDR1E JSR  LCNAB3  FORM MACHINE CODE
OE70 20 19      BRA  ADDR1J
*
OE72 C6 90 ADDR1F LDA  B #S90    DIRECT, FORM A
OE74 F7 1087 R      STA  B ORBYA  NIBBLE
OE77 C6 D0      LDA  B #SD0     OF
OE79 F7 1088 R      STA  B ORBYB  MACHINE CODE
OE7C 20 0A      BRA  ADDR1H
*
OE7E C6 A0 ADDR1G LDA  B #SA0    INDEXED, FORM A
OE80 F7 1087 R      STA  B ORBYA  NIBBLE OF
OE83 C6 E0      LDA  B #SEQ     OF
OE85 F7 1088 R      STA  B ORBYB  MACHINE CODE
*
OE86 B0 1113 R ADDR1H JSR  LCNAB2  FORM MACHINE CODE
OE88 B0 11C2 R ADDR1J JSR  LCLCN   LC*=LC+LCN
OE8E 7E 0490 R      JMP  MAIN1   RETURN TO MAIN LOOP
*
* **ADDRESS TYPE 2**
*
* [SIA]
*
* DIRECT (2 BYTES) OR EXTENDED (3 BYTES)

```

```

* CCC A NUMBER          CCC B NUMBER
* CCC A SYMBOL          CCC B SYMBOL
* CCC A EXPRESSION      CCC B EXPRESSION
*
* INDEXED(2 BYTES):
* CCC A NUMBER,X        CCC B NUMBER,X
* CCC A SYMBOL,X        CCC B SYMBOL,X
* CCC A EXPRESSION,X    CCC B EXPRESSION,X
*
OE91 BD 1089 R ADDR2 JSR  ADRINT  INIT ADDRESS FIELD FLAGS
OE94 BD 06F6 R      JSR  NX10K    GET NEXT TOKEN
OE97 C1 0D      CMP  B  #50J     EOL?
OE99 26 08      BNE  ADDR2B      NO
*
OE9B CE 0204      ADDR2A LDX  #50204 ERROR
OE9E BD 0DBB R    JSR  PRINTE    PRINT
OEAT 20 32      BRA  ADDR2E      RETURN
*
OEAB BD 10B7 R ADDR2B JSR  ABRCK   CHECK FOR REGISTER A OR B
OEAO F6 1084 R    LDA  B  ABR     NEITHER?
OEAY 27 F0      BEQ  ADDR2A      YES ERROR
OEAB BD 06F6 R    JSR  NX10K     GET NEXT TOKEN
OEAE BD 09B5 R    JSR  NSEVL     EVALUATE OPERAND
OEB1 BD 10J7 R    JSR  P2ERR     PRINT PASS 2 ERRORS
OEB4 BD 06F6 R    JSR  NX10K     GET NEXT TOKEN
OEB7 BD 10C4 R    JSR  INXCK     INDEXED?
OEBA 26 2A      BNE  ADDR2G      YES
*
OEBc 7D 0278 R    IST  CMNFLG    COMMON?
OEBF 26 0A      BNE  ADDR2K      YES
*
OEC1 7D 0277 R    IST  RELFLG    RELOC ?
OEC4 26 05      BNE  ADDR2K      YES
*
OEC6 F6 0C68 R    LDA  B  ADR1    DIRECT?
OEC9 27 0F      BEQ  ADDR2F      YES
*
OECB C6 B0      ADDR2K LDA  B  #5B0 EXTENDED, FORM A
OECJ F7 1087 R   STA  B  ORBYA     NIBBLE
OED0 C6 F0      LDA  B  #5F0      OF
OED2 F7 1088 R   STA  B  ORBYB     MACHINE CODE
*
OED5 BD 115E R ADDR2E JSR  LCNAB3  FORM MACHINE CODE
OED8 20 19      BRA  ADDR2J
*
OEDA C6 90      ADDR2F LDA  B  #590 DIRECT, FORM A
OEDC F7 1087 R   STA  B  ORBYA     NIBBLE
OEDF C6 00      LDA  B  #5D0      OF
OEE1 F7 1088 R   STA  B  ORBYB     MACHINE CODE
OEE4 20 0A      BRA  ADDR2H
*
OEE6 C6 A0      ADDR2G LDA  B  #5A0 INDEXED, FORM A
OEE8 F7 1087 R   STA  B  ORBYA     NIBBLE
OEEB C6 E0      LDA  B  #5E0      OF
OEEF F7 1035 R   STA  B  ORBYB     MACHINE CODE
*
OEF0 BD 1113 R ADDR2H JSR  LCNAB2  FORM MACHINE CODE
OEF3 BD 11C2 R ADDR2J JSR  LCLCN    LC:=LC+LCN
OEF6 7E 0490 R   JMP  MAIN1      RETURN TO MAIN LOOP
*
**ADDRESS TYPE 3**
*
* [ASL ASR CLR COM DEC INC LSR NEG ROL ROR TST]
*
* ACCUMULATOR(1 BYTE):
* CCC A
* CCC B
*
* EXTENDED(3 BYTES):
* CCC NUMBER
* CCC SYMBOL
* CCC EXPRESSION
*
* INDEXED(2 BYTES)
* CCC NUMBER,X
* CCC SYMBOL,X
* CCC EXPRESSION,X
*
*
OEF9 BD 1089 R ADDR3 JSR  ADRINT  INIT ADDRESS FIELD FLAGS
OEFc BD 06F6 R      JSR  NX10K    GET NEXT TOKEN
OEFF C1 0D      CMP  B  #50J     EOL?
OF01 26 08      BNE  ADDR3B      NO
*
OF03 CE 0204      LDX  #50204 ERROR
OF06 BD 0DBB R    JSR  PRINTE    PRINT
OF09 20 2A      BRA  ADDR3D      RETURN
*
OF0B BD 10B7 R ADDR3B JSR  ABRCK   CHECK FOR REGISTER A OR B
OF0E 7D 1084 R    IST  ABR     NEITHER?
OF11 27 0F      BEQ  ADDR3C      YES

```

```

*
0F13 C6 40          LDA B #$40      ACCUMULATOR, FORM A
0F15 F7 1087 R      STA B ORBYA     NIBBLE
0F18 C6 50          LDA B #$50      OF
0F1A F7 1088 R      STA B ORBYB     MACHINE CODE
0F1D BD 10EA R      JSR LCNAB1      FORM MACHINE CODE
0F20 20 20          BRA ADDR3F

*
0F22 BD 09B5 R ADDR3C JSR NSEVL     EVALUATE OPERAND
0F25 BD 10D7 R      JSR P2ERR      PRINT PASS 2 ERRORS
0F28 BD 06F6 R      JSR NXTOK      GET NEXT TOKEN
0F2E BD 10C4 R      JSR INXCK      INDEXED?
0F2E 26 0A          BNE ADDR3E      YES

*
0F30 C6 70          LDA B #$70      EXTENDED, FORM A
0F32 F7 1087 R      STA B ORBYA     NIBBLE OF MACHINE CODE

*
0F35 BD 117C R ADDR3D JSR LCN3      FORM MACHINE CODE
0F38 20 08          BRA ADDR3F

*
0F3A C6 60          ADDR3E LDA B #$60 INDEXED, FORM A
0F3C F7 1087 R      STA B ORBYA     NIBBLE OF MACHINE CODE
0F3F BD 1131 R      JSR LCN2       FORM MACHINE CODE

*
0F42 BD 11C2 R ADDR3F JSR LCLCN     LC*=LC+LCN
0F45 7E 0490 R      JMP MAIN1      RETURN TO MAIN LOOP

*
* **ADDRESS TYPE 4**
*
* [PSH PUL]
*
* ACCUMULATOR (1 BYTE):
*   PSH A
*   PSH B
*   PUL A
*   PUL B
*
*
0F48 BD 1089 R ADDR4  JSR ADRINT    INIT ADDRESS FIELD FLAGS
0F4B BD 06F6 R      JSR NXTOK      GET NEXT TOKEN
0F4E BD 1087 R      JSR ABRCK      CHECK FOR A,B REGS
0F51 7D 1084 R      1ST ABR        NEITHER ?
0F54 26 06          BNE ADDR4A     NO

*
0F56 CE 0204          LDX #$0204    ERROR
0F59 BD 0D8B R      JSR PRINTE

*
0F5C 7C 1088 R ADDR4A INC ORBYB     ORBYB*=01
0F5F BD 10EA R      JSR LCNAB1     FORM MC
0F62 BD 11C2 R      JSR LCLCN     LC*=LC+LCN
0F65 7E 0490 R      JMP MAIN1      RETURN TO MAIN LOOP

*
* **ADDRESS TYPE 5**
*
* [CPX LDS LDX]
*
* IMMEDIATE(3 BYTES):
*   CCC #NUMBER
*   CCC #SYMBOL
*   CCC #EXPRESSION
*   CCC #'CC
*
* DIRECT(2 BYTES) OR EXTENDED(3 BYTES):
*   CCC NUMBER
*   CCC SYMBOL
*   CCC EXPRESSION
*
* INDEXED(2 BYTES)
*   CCC NUMBER,X
*   CCC SYMBOL,X
*   CCC EXPRESSION,X
*
*
0F68 BD 1089 R ADDR5  JSR ADRINT    INIT ADDRESS FIELD FLAGS
0F6B BD 06F6 R      JSR NXTOK      GET NEXT TOKEN
0F6E C1 00          CMP B #$00      EOL?
0F70 26 08          BNE ADDR5B     NO

*
0F72 CE 0240 ADDR5A LDX #$0240    ERROR
0F75 BD 0D8B R      JSR PRINTE     PRINT
0F78 20 46          BRA ADDR5E     RETURN

*
0F7A C1 23          ADDR5B CMP B #$23 IMMEDIATE?
0F7C 26 19          BNE ADDR5C     NO
0F7E 73 1085 R      COM IMMED      SET IMMEDIATE FLAG
0F81 BD 06F6 R      JSR NXTOK      GET NEXT TOKEN
0F84 C1 27          CMP B #$27     " " ?
0F88 26 0F          BNE ADDR5C     NO

*
0F88 FE 027E R      LDX CUCHAR     YES, GET NEXT TWO CHARS
0F8B A6 00          LDA A 0,X
0F8D B7 0C68 R      STA A ADR1
0F90 A6 01          LDA A 1,X
0F92 B7 0C69 R      STA A ADR2

```



```

0F95 20 29          BRA    ADDR5E

*
0F97 BD 09B5 R ADDR5C JSR    NSEVL    EVALUATE OPERAND
0F9A BD 1007 R      JSR    P2ERR    PRINT PASS 2 ERRORS
0F9D F6 10B5 R      LDA B IMMED    IMMEDIATE?
0FA0 27 02          BEQ    ADDR5D    NO
0FA2 20 1C          BRA    ADDR5E    YES

*
0FA4 BD 06F6 R ADDR5D JSR    NXTOK    GET NEXT TOKEN
0FA7 BD 10C4 R      JSR    INXCK    INDEXED?
0FAA 26 20          BNE    ADDR5G    YES

*
0FAC 7D 0278 R      TST    CMNFLG    COMMON?
0FAF 26 0A          BNE    ADDR5K    YES

*
0FB1 7D 0277 R      TST    RELFLG    RELOC ?
0FB4 26 05          BNE    ADDR5K    YES

*
0FB6 F6 0C08 R      LDA B ADRI     DIRECT?
0FB9 27 0A          BEQ    ADDR5F    YES

*
0FBb C6 30          ADDR5K LDA B #S30    EXTENDED,FORM A
0FBd F7 10d7 R      STA B ORBYA    NIBBLE OF MACHINE CODE
0FC0 BD 117C R ADDR5E JSR    LCN3     FORM MACHINE CODE
0FC3 20 0F          BRA    ADDR5J

*
0FC5 C6 10          ADDR5F LDA B #S10    DIRECT,FORM A
0FC7 F7 10d7 R      STA B ORBYA    NIBBLE OF MACHINE CODE
0FCA 2J 05          BRA    ADDR5H

*
0FCC C6 20          ADDR5G LDA B #S20    INDEXED,FORM A
0FCE F7 10d7 R      STA B ORBYA    NIBBLE OF MC

*
0FD1 BD 1131 R ADDR5H JSR    LCN2     FORM MC
0FD4 BD 11C2 R ADDR5J JSR    LCLCN    LC:=LC+LCN
0FD7 7E 0490 R      JMP    MAIN1    RETURN TO MAIN LOOP

*          **ADDRESS TYPE 6**
*
* [STX,STS]
*
* DIRECT (2 BYTES) OR EXTENDED(3 BYTES):
*   CCC NUMBER
*   CCC SYMBOL
*   CCC EXPRESSION
*
* INDEXED (2 BYTES):
*   CCC NUMBER,X
*   CCC SYMBOL,X
*   CCC EXPRESSION,X
*
0FDA BD 10B9 R ADDR6  JSR    ADRINT    INIT ADDRESS FIELD FLAGS
0FDd BD 00F6 R      JSR    NXTOK    GET NEXT TOKEN
0FEO C1 0D          CMP B #S0D     EOL?
0FE2 26 B3          BNE    ADDR5C    NO
0FE4 20 8C          BRA    ADDR5A    YES,ERROR

*          **ADDRESS TYPE 7**
*
* [JMP JSR]
*
*
* INDEXED (2 BYTES):
*   CCC NUMBER,X
*   CCC SYMBOL,X
*   CCC EXPRESSION,X
*
0FE6 BD 10B9 R ADDR7  JSR    ADRINT    INIT ADDRESS FIELD FLAGS
0FE9 BD 00F6 R      JSR    NXTOK    GET NEXT TOKEN
0FEC C1 0D          CMP B #S0D     EOL?
0FEE 26 08          BNE    ADDR7A    NO

*
0FF0 CE 0204        LUX    #S0204    ERROR
0FF3 BD 0D8B R      JSR    PRINTE    PRINT
0FF6 2J 0E          BRA    ADDR7B

*
0FF8 BD 09B5 R ADDR7A JSR    NSEVL    EVALUATE OPERAND
0FFB BD 1007 R      JSR    P2ERR    PRINT PASS 2 ERRORS
0FFE BD 06F6 R      JSR    NXTOK    GET NEXT TOKEN
1001 BD 10C4 R      JSR    INXCK    INDEXED?
1004 26 0A          BNE    ADDR7C    YES

*
1006 C6 10          ADDR7B LDA B #S10    EXTENDED,FORM A NIBBLE
1008 F7 10d7 R      STA B ORBYA    OF MC
100b BD 117C R      JSR    LCN3     FORM MACHINE CODE
100E 20 03          BRA    ADDR7D

*
1010 BD 1131 R ADDR7C JSR    LCN2     FORM MACHINE CODE
1013 BD 11C2 R ADDR7D JSR    LCLCN    LC:=LC+LCN
1016 7E 0490 R      JMP    MAIN1    RETURN TO MAIN LOOP

*          **ADDRESS TYPE 8**
*
* [BCC BCS BEQ BGE BGT BHI BLE BLS

```

```

* BLT BMI BNE BPL BRA BSR BVC BVSJ
*
* RELATIVE (2 BYTES):
*   CCC   NUMBER
*   CCC   SYMBOL
*   CCC   EXPRESSION
*
1019 BD 1089 R ADDR8 JSR  ADRINT  INIT ADDRESS FIELD FLAGS
101C BD 06F6 R      JSR  NXTOK   GET NEXT TOKEN
101F C1 0D      CMP  B  #0D     EOL?
1021 26 08      BNE  ADDR8A    NO
*
1023 CE 0204      LDX  #0204    ERROR
1026 BD 0D8B R      JSR  PRINTE  PRINT
1029 20 36      BRA  ADDR8D
*
102B 7D 0275 R ADDR8A 1ST  PASS  ?
102E 27 31      BEQ  ADDR8D    PASS1
*
1030 BD 09B5 R      JSR  NSEVL   PASS 2 EVAL OPERAND
1033 BD 10D7 R      JSR  P2ERR   PRINT PASS 2 ERRORS
1036 FE 0273 R      LDX  LC      LSAVE:=LC+2
1039 08      INX
103A 08      INX
103B FF 0285 R      STX  LSAVE
103E B6 0C69 R      LDA  A  ADR2  CALCULATE OFFSET
1041 F6 0C68 R      LDA  B  ADR1
1044 B0 0286 R      SUB  A  LSAVE+1
1047 F2 0285 R      SBC  B  LSAVE
*
104A C1 FF      CMP  B  #$FF    CHECK FOR OUT OF RANGE
104C 26 03      BNE  ADDR8E
104E 4D      TST  A
104F 2B 0D      BMI  ADDR8C    NEGATIVE? (FF - 80)
                                OK
*
1051 C1 00      ADDR8E CMP  B  #00
1053 26 03      BNE  ADDR8F
1055 4D      TST  A
1056 2A 06      BPL  ADDR8C    OUT OF RANGE
                                POSITIVE? (00 - 7F)
                                OK
*
1058 CE 0208      ADDR8F LDX  #0208  ERROR
105B BD 0D8B R      JSR  PRINTE  PRINT
*
105E B7 0C69 R ADDR8C STA  A  ADR2  SAVE OFFSET
1061 BD 1131 R ADDR8D JSR  LCN2   FORM MC
1064 BD 11C2 R      JSR  LCLCN   LC:=LC+LCN
1067 7E 0490 R      JMP  MAIN1   RETURN TO MAIN LOOP
*
* ADDRESS TYPE ***
*
* LABA CBA CLC CLI CLV DES DEX INS
* INX NOP RTI RTS SBA SEC SEI SEV
* SWI TAB TAP TBA TPA TSX TXS WAI
*
* INHERENT(1 BYTE):
*   CCC
*
106A BD 1089 R ADDR9 JSR  ADRINT  INIT ADDRESS FIELD FLAGS
106D 7D 0275 R      1ST  PASS  ?
1070 27 03      BEQ  ADDR9A    PASS 1
*
1072 BD 102E R      JSR  OUTBIN  OUTPUT MC
*
1075 7C 0C65 R ADDR9A INC  MCOUNT MCOUNT:=1
1076 7C 0234 R      INC  LCN     LCN:=1
107C BD 0C0E R      JSR  PRINTL
107E BD 11C2 R      JSR  LCLCN   LC:=LC+LCN
1081 7E 0490 R      JMP  MAIN1   RETURN TO MAIN LOOP
*
* ROUTINES USED TO INIT AND CHECK ADDRESS FIELD
* FLAGS, MC FORMS AND LISTING FLAGS.
*
1084 0001      ABR  RMB  1      REG A OR B FLAG
1085 0001      IMMED RMB  1      IMMEDIATE MODE FLAG
1086 0001      INDEX RMB  1      INDEX MODE FLAG
1087 0001      ORBYA RMB  1      FORM FOR A NIBBLE OF MC
1088 0001      ORBYB RMB  1      FORM FOR A NIBBLE OF MC
*
1089 7F 0234 R ADRINT CLR  LCN
108C 7F 0277 R      CLR  RELFLG
108F 7F 0276 R      CLR  CMNFLG
1092 7F 0279 R      CLR  EXIFLG
1095 7F 027A R      CLR  ENIFLG
1098 7F 0C66 R      CLR  POP
109B F7 0C67 R      STA  B  OPCD  SAVE OPCODE
109E 7F 0C65 R      CLR  MCOUNT
10A1 7F 1084 R      CLR  ABR
10A4 7F 1085 R      CLR  IMMED
10A7 7F 1086 R      CLR  INDEX
10AA 7F 0C68 R      CLR  ADR1
10AD 7F 0C69 R      CLR  ADR2

```

```

10B0 7F 1087 R      CLR  ORBYA
10B3 7F 1088 R      CLR  ORBYB
10B6 39             RTS

*
*
* CHECK FOR PRESENCE OF A OR B REG
*
10B7 C1 41      ABRCK  CMP B  #541      "A" ?
10B9 27 05      BEQ   ABRCKA      YES
10BB C1 42      CMP B  #542      "B" ?
10BD 27 01      BEQ   ABRCKA      YES
10BF 39             RTS          NEITHER, RETURN

*
10C0 F7 1084 R  ABRCKA STA B ABR      SAVE REG
10C3 39             RTS

*
*
* CHECK FOR INDEXED MODE
*
10C4 C1 2C      INXCK  CMP B  #52C      ", " ?
10C6 26 0B      BNE   INXCKR      NO
10C8 BD 06F6 R  JSR    NX10K      GET NEXT TOKEN
10CB C1 58      CMP B  #558      "X" ?
10CD 26 04      BNE   INXCKR      NO
10CF 73 1086 R  COM    INDEX      INDEX==FF
10D2 39             RTS

*
10D3 7F 1086 R  INXCKR CLR  INDEX
10D6 39             RTS

*
*
* CHECK FOR PASS 2 ERRORS
*
10D7 C1 FF      P2ERR  CMP B  #5FF      ERROR (FROM NSEVL)?
10D9 26 0E      BNE   P2ERRB      NO

*
10DB 7D 0275 R  1ST    PASS      YES,PASS?
10DE 27 03      BEQ   P2ERRA      PASS1
10E0 BD 0DBB R  JSR    PRINTE      PASS 2,PRINT ERROR
10E3 7F 0C68 R  P2ERRA CLR  ADRI
10E6 73 0C68 R  COM    ADRI      ADRI==FF (TO KILL DIRECT)
10E9 39             P2ERRB RTS

* ROUTINES TO FINISH UP ADDRESS TYPE PROCESSING
* THESE ROUTINES DO THE FOLLOWING:
*   PASS 1
*   A. LCN== # OF BYTES IN THE INSTRUCTION
*   PASS 2
*   A. FORM COMPLETE OPCODE
*   B. OUTPUT MACHINE CODE GENERATED
*   C. PRINT A LINE OF LISTING
*   D. LCN== # OF BYTES IN THE INSTRUCTION
*
* LCNAB1 1 BYTE ACCUMULATOR INSTRUCTIONS
*
10EA 7D 0275 R  LCNAB1 1ST    PASS      PASS?
10ED 27 20      BEQ   LCNABIS      PASS 1

*
10EF F6 0C67 R  LDA B OPCD      PASS 2,LOAD PARTIAL OPCODE

* EXTENDED (3 BYTES):
*   CCC NUMBER
*   CCC SYMBOL
*   CCC EXPRESSION
10F2 B6 1084 R  LDA A ABR      A OR B ?
10F5 27 0F      BEQ   LCNAB10     NEITHER

*
10F7 81 42      CMP A  #542      "B" ?
10F9 27 05      BEQ   LCNAB1B     YES
10FB FA 1087 R  ORA B ORBYA      A FORM COMPLETE OPCODE
10FE 20 03      BRA   LCNAB1C

*
1100 FA 1088 R  LCNAB1B ORA B ORBYB  B FORM COMPLETE OPCODE
1103 F7 0C67 R  LCNAB1C STA B OPCD  SAVE
1106 BD 182E R  LCNAB10 JSR    OUTBIN  OUTPUT OPCODE
1109 7C 0C65 R  INC    MCOUNT  MCOUNT==1
110C BD 0C0E R  JSR    PRINTL     PRINT A LINE OF LISTING
110F 7C 0284 R  LCNAB1S INC    LCN   LCN=1
1112 39             RTS          RETURN

*
*
* LCNAB2 2 BYTE REGISTER (A,B);INDEXED,
*   DIRECT, AND IMMEDIATE TYPE INSTRUCTIONS
*
1113 7D 0275 R  LCNAB2 1ST    PASS      PASS ?
1116 27 3F      BEQ   LCN2B      PASS 1

*
1118 F6 0C67 R  LDA B OPCD      PASS 2,GET PARTIAL OPCODE
111B B6 1084 R  LDA A ABR      A OR B ?
111E 27 25      BEQ   LCN2A      NEITHER

*
1120 81 42      CMP A  #542      B ?
1122 27 05      BEQ   LNB2       YES

```

```

1124 FA 1087 R      ORA B ORBYA    A, FORM COMPLETE OPCODE
1127 20 03          BRA    LNAB2S

*
1129 FA 1088 R LNB2  ORA B ORBYB    B, FORM COMPLETE OPCODE
112C F7 0C67 R LNAB2S STA B OPCD    SAVE
112F 20 14          BRA    LCN2A     FINISH UP

*
* LCN2 2 BYTE INDEXED,DIRECT,AND IMMEDIATE TYPE
* INSTRUCTIONS
*
1131 7D 0275 R LCN2  IST    PASS    PASS ?
1134 27 21          BEQ    LCN2B     PASS 1

*
1136 7F 0277 R      CLR    RELFLG
1139 7F 0278 R      CLR    CMNFLG
113C F6 0C67 R      LDA B OPCD    PASS 2,GET PARTIAL OPCODE
113F FA 1087 R      ORA B ORBYA    FORM COMPLETE OPCODE
1142 F7 0C67 R      STA B OPCD    SAVE

*
1145 BD 182E R LCN2A JSR    OUTBIN    OUTPUT OPCODE
1148 F6 0C69 R      LDA B ADR2     GET ADDRESS PART OF MC
114B BD 182E R      JSR    OUTBIN    OUTPUT IT
114E 7C 0C65 R      INC    MCOUNT MCOUNT+=2
1151 7C 0C65 R      INC    MCOUNT
1154 BD 0C0E R      JSR    PRINTL    PRINT A LINE OF LISTING

*
1157 7C 0284 R LCN2B INC    LCN      LCN+=2
115A 7C 0284 R      INC    LCN
115D 39            RTS             RETURN

*
* LCNAB3 3 BYTE REGISTER(A,B);EXTENDED TYPE
* INSTRUCTIONS
*
115E 7D 0275 R LCNAB3 IST    PASS    PASS ?
1161 27 55          BEQ    LCN3B     PASS 1

*
1163 F6 0C67 R      LDA B OPCD    PASS 2 GET PARTIAL OPCODE
1166 B6 1034 R      LDA A ABR      A OR B ?
1169 27 1F          BEQ    LCN3A     NEITHER
116B 81 42          CMP A #542     B ?
116D 27 05          BEQ    LNB3     YES
116F FA 1087 R      ORA B ORBYA    A, FORM COMPLETE OPCODE
1172 20 03          BRA    LNAB3S

*
1174 FA 1088 R LNB3  ORA B ORBYB    B, FORM COMPLETE OPCODE
1177 F7 0C67 R LNAB3S STA B OPCD    SAVE
117A 20 0E          BRA    LCN3A     FINISH UP

*
* LCN3 3 BYTE EXTENDED AND IMMEDIATE TYPE
* INSTRUCTIONS
*
117C 7D 0275 R LCN3  IST    PASS    PASS ?
117F 27 37          BEQ    LCN3B     PASS 1

*
1181 F6 0C67 R      LDA B OPCD    GET PARTIAL OPCODE
1184 FA 1087 R      ORA B ORBYA    FORM COMPLETE OPCODE
1187 F7 0C67 R      STA B OPCD    SAVE

*
118A BD 182E R LCN3A JSR    OUTBIN    OUTPUT OPCODE
118D F6 0C68 R      LDA B ADR1     OUTPUT THE REST OF THE MC
1190 BD 182E R      JSR    OUTBIN
1193 F6 0C69 R      LDA B ADR2
1196 BD 182E R      JSR    OUTBIN

*
1199 7D 0278 R      IST    CMNFLG    COMMON?
119C 27 04          BEQ    **+6     NO

*
119E C6 4D          LDA B #7M      "COMMON"
11A0 20 07          BRA    LCN3C

*
11A2 7D 0277 R      IST    RELFLG    RELOC ?
11A5 27 05          BEQ    **+7     NO

*
11A7 C6 52          LDA B #522     LOAD "R"
11A9 BD 1842 R LCN3C JSR    OUTBNR

*
11AC 7C 0C65 R      INC    MCOUNT MCOUNT+=3
11AF 7C 0C65 R      INC    MCOUNT
11B2 7C 0C65 R      INC    MCOUNT
11B5 BD 0C0E R      JSR    PRINTL    PRINT A LINE OF LISTING

*
11B8 7C 0284 R LCN3B INC    LCN      LCN+=3
11BB 7C 0284 R      INC    LCN
11BE 7C 0284 R      INC    LCN
11C1 39            RTS             RETURN

*
* LCLCN LC*=LC+LCN
*

```

```

11C2 B6 0274 R LCLCN LDA A LC+1
11C5 F6 0273 R LDA B LC
11C8 B8 0264 R ADD A LCN ADD LCN
11CB C9 00 ADC B #500
11CD B7 0274 R STA A LC+1 SAVE LC
11D0 F7 0273 R STA B LC
11D3 39 RTS RETURN

* POCMN: ALLOCATE COMMON STORAGE AREAS
*
11D4 BD 1039 R POCAN JSR ADRINT
11D7 BD 151A R JSR LBLCK
*
11DA BD 06F6 R JSR NXTOK GET SYMBOL NAME
11DD C1 01 CMP B #1 OK?
11DF 27 08 BEQ POCMN2 YES
*
11E1 3E 0216 POCAN LDX #0216 ERROR
11E4 BD 0DBB R POCAN1 JSR PRINT
11E7 29 5J BRA POCAN4
*
11E9 7D 0275 R POCAN2 IST PASS PASS ?
11EC 26 41 BNE POCMN3 PASS 2
*
11EE BD 07F8 R JSR STOSYM ENTER NAME IN SYMTAB
11F1 FE 0282 R LDX SYMPTR
11F4 FF 124C R STX CMNXS SAVE ENTRY ADDRESS
*
11F7 BD 06F6 R JSR NXTOK GET DELIM.
11FA C1 2C CMP B #52C " " ?
11FC 26 E3 BNE POCMN0 NO
*
11FE BD 06F6 R JSR NXTOK POINT TO OPERAND
1201 BD 0985 R JSR NSEVL GET VALUE
1204 C1 FF CMP B #5FF OK?
1206 27 DC BEQ POCMN1 NO
*
1208 FE 124C R LDX CMNXS POINT TO ENTRY
120B 86 BF LDA A #5BF
120D A4 08 AND A 8,X TURN OFF REL BIT
120F 8A 10 ORA A #10 TURN ON COMMON BIT
1211 A7 08 STA A 8,X
*
1213 B6 0313 R LDA A CMNLC GET COMMON LC
1216 A7 06 STA A 6,X STORE IN ENTRY
1218 B6 0314 R LDA A CMNLC+1
121B A7 07 STA A 7,X
*
* CMNLC:=CMNLC+[ADR1,ADR2]
*
121D B6 0C69 R LDA A ADR2
1220 F6 0C68 R LDA B ADR1
1223 B8 0314 R ADD A CMNLC+1
1226 F9 0313 R ADC B CMNLC
1229 B7 0314 R STA A CMNLC+1
122C F7 0313 R STA B CMNLC
*
122F BD 0857 R POCMN3 JSR LKPSYM LOOK UP SYMBOL
1232 FE 0282 R LDX SYMPTR POINT TO ENTRY
1235 EE 06 LDX 6,X GET COMMON ADDRESS
1237 FF 0C68 R STX ADR1 SET UP FOR PRINTL
123A 73 0C66 R COM POP
123D 73 0278 R COM CMNFLG
1240 7C 0C65 R INC MCOUNT
1243 7C 0C65 R INC MCOUNT
*
1246 BD 0C0E R POCMN4 JSR PRINIL
1249 7E 0490 R JMP MAINI
*
124C 0002 CMNXS RMB 2
* POEND: PROCESS END PSEUDOP
*
124E BD 1039 R POEND JSR ADRINT INIT FLAGS
1251 BD 151A R JSR LBLCK CHECK FOR A LABEL
1254 7D 0275 R POEND0 IST PASS PASS ?
1257 26 0F BNE POEND2 PASS2
*
1259 FE 0273 R LDX LC PASS1
125C FF 0271 R STX 151PH 151PH:=LC
125F 73 0275 R COM PASS PASS:=PASS2
*
1262 BD 025F R JSR RESTR REWIND INPUT FILE
1265 7E 0467 R JMP PASS2 EXECUTE PASS2
*
1268 FE 0271 R POEND2 LDX 151PH PHASING ERRORS?
126B BC 0273 R CPX LC
126E 27 06 BEQ ENDP2 NO
*
1270 CE 0220 LDX #0220
1273 BD 0DBB R JSR PRINT PRINT ERROR
*

```

1276 B6 026E R ENDP2	LDA A 0PINS.	
1279 85 80	BIT A #580	LISTING?
127B 26 06	BNE ENDP3	NO
	*	
127D BD 0C0E R	JSR PRINTL	
1280 BD 13BE R	JSR CRLF	
	*	
1283 B6 026E R ENDP3	LDA A 0PINS	
1286 85 20	BIT A #520	LIST SYMTAB?
1288 27 03	BEQ SORT1	YES
	*	
128A 7E 134D R	JMP ENDP6	NO
	*	
128D CE 13C8 R SORT1	LDX #ZZZ	INIT SORT
1290 FF 13D1 R	STX CBLOCK	
1293 7F 13D5 R	CLR SORTF	CLEAR SORT FLAG
1296 FE 0268 R	LDX SYMTAB	POINT TO TABLE
1299 20 09	BRA SORT3	
	*	
129B 08	SORT2 INX	
129C 08	INX	
129D 08	INX	
129E 08	INX	
129F 08	INX	
12A0 08	INX	
12A1 08	INX	
12A2 08	INX	
12A3 08	INX	
	*	
12A4 BC 026C R SORT3	CPX SYMEND	AT TABLE END?
12A7 26 0B	BNE SORT2A	NO
	*	
12A9 7D 13D5 R	IST SORTF	FOUND AN ENTRY?
12AC 27 03	BEQ **5	
	*	
12AE 7E 12EB R	JMP SORT5	PRINT ENTRY
12B1 7E 134D R	JMP ENDP6	ALL DONE
	*	
12B4 E6 00	SORT2A LDA B 0,X	
12B6 C1 20	CMP B #520	BLANK?
12B8 27 E1	BEQ SORT2	YES, GET NEXT ENTRY
	*	
12BA E6 08	LDA B 8,X	
12BC C1 FF	CMP B #5FF	USED ENTRY?
12BE 27 0B	BEQ SORT2	YES
	*	
	* COMPARE ENTRY AT CBLOCK WITH NEW ENTRY	
	*	
12C0 FF 13D3 R	STX CXS2	SET UP FOR COMPARISON
12C3 FF 07E5 R	STX PSTNG2	
12C6 FE 13D1 R	LDX CBLOCK	
12C9 FF 07E3 R	STX PSTNG1	
12CC C6 06	LDA B #6	
12CE F7 07E7 R	STA B PCOUNT	
12D1 CE 07E3 R	LDX #PSTNG1	
12D4 BD 06C5 R	JSR COMPAR	
12D7 22 05	BHI SORT4	NEED SWITCH
	*	
12D9 FE 13D3 R	LDX CXS2	
12DC 20 BD	BRA SORT2	
	*	
12DE FE 13D3 R SORT4	LDX CXS2	NEW CBLOCK PTRS
12E1 FF 13D1 R	STX CBLOCK	
12E4 C6 FF	LDA B #5FF	
12E6 F7 13D5 R	STA B SORTF	SET SORT FLAG
12E9 20 80	BRA SORT2	
	*	
12EB BD 0C2A R SORT5	JSR LINCK	
12EE C6 06	LDA B #6	
12F0 FE 13D1 R	LDX CBLOCK	
	*	
12F3 A0 00	ENDP4 LDA A 0,X	GET CHAR
12F5 BD 1888 R	JSR OUTCHR	PRINT
12F8 08	INX	POINT TO NEXT CHAR
12F9 5A	DEC B	DECREMENT COUNT
12FA 26 F7	BNE ENDP4	NOT DONE
	*	
12FC 86 20	LDA A #520	PRINT BLANK
12FE BD 1888 R	JSR OUTCHR	
1301 BD 1871 R	JSR OUT4HS	PRINT 4 HEX LOCATION
1304 FF 13D6 R	STX ENDXS	
1307 E6 00	LDA B 0,X	
1309 C5 40	BIT B #540	RELOC ?
130B 27 05	BEQ **7	NO
	*	
130D 86 52	LDA A #552	LOAD "R"
130F BD 1888 R	JSR OUTCHR	PRINT IT
	*	
1312 C5 20	BIT B #520	MACRO NAME?
1314 27 05	BEQ **7	NO
	*	
1316 86 4D	LDA A #*M	LOAD M

```

1318 BD 1888 R      JSR  OUTCHR      PRINT IT
*
131B C5 10          BIT  B #$10      COMMON?
131D 27 05          BEQ  **7         NO
*
131F 86 43          LDA  A #'C
1321 BD 1888 R      JSR  OUTCHR
*
1324 C5 08          BIT  B #$08      EXTERNAL?
1326 27 05          BEQ  **7         NO
*
1328 86 58          LDA  A #'X
132A BD 1888 R      JSR  OUTCHR
*
132D C5 04          BIT  B #$04      ENTRY?
132F 27 05          BEQ  **7
*
1331 86 4E          LDA  A #'N
1333 BD 1888 R      JSR  OUTCHR
*
1336 E6 00          LDA  B 0,X        REDEFINED?
1338 2A 06          BPL  ENDP5        NO
*
133A CE 138A R      LDX  #REDEF      PRINT ERROR MESSAGE
133D BD 1861 R      JSR  PDATA1
*
1340 FE 13D6 R      ENDP5            LDX  ENDXS
1343 C6 FF          LDA  B #$FF      SET DONE
1345 E7 00          STA  B 0,X
1347 BD 13BE R      JSR  CRLF
134A 7E 128D R      JMP  SORT1
*
134D BD 13BE R      ENDP6            JSR  CRLF
1350 BD 13BE R      JSR  CRLF
1353 CE 13A1 R      LDX  #ENDMB      PRINT # OF ERRORS MSG
1356 B6 0288 R      LDA  A ECOUNT
1359 F6 0289 R      LDA  B ECOUNT+1
135C BD 0D5C R      JSR  CVBTD      CONVERT TO ASCII
135F CE 1395 R      LDX  #ENDMA
1362 BD 1861 R      JSR  PDATA1      PRINT IT
*
1365 BD 13BE R      JSR  CRLF
1368 BD 13BE R      JSR  CRLF
*
136B CE 13AE R      LDX  #CMMSG      COMMON AREA MESSAGE
136E BD 1861 R      JSR  PDATA1
*
1371 CE 0313 R      LDX  #CMNLC      POINT TO COMMON LENGTH
1374 BD 1871 R      JSR  OUT4HS
1377 BD 13BE R      JSR  CRLF
*
137A 86 026E R      LDA  A OPINS
137D 85 40          BIT  A #$40      OBJECT?
137F 26 06          BNE  ENDP7        NO
*
1381 BD 0259 R      ENDP6A           JSR  WEOF      WRITE EOF NULLS
1384 7E 024D R      JMP  UPDATE      CLOSE FILE AND EXIT TO MONITOR
*
1387 7E 0250 R      ENDP7            JMP  MONITOR
*
133A 20             REDEF  FCC  'REDEFINED'
1394 04             FCB  $04      EOT
*
1395 54             ENDMA  FCC  'THERE WERE: '
13A1 0005           ENDMB  RMB  5
13A6 20             FCC  ' ERRORS'
13AD 04             FCB  4
*
13AE 43             CMMSG  FCC  'COMMON LENGTH= '
13BD 04             FCB  4
*
13BE CE 13C5 R      CRLF  LDX  #MCRLF
13C1 BD 1861 R      JSR  PDATA1
13C4 39             RTS
*
13C5 000A           MCRLF  FDB  $000A      CR,LF
13C7 04             FCB  $04      EOT
*
13C8 58             ZZZ  FCC  '[[[[[[[
13CE 0000           FDB  0000
13D0 00             FCB  0
13D1 0002           CBLOCK RMB 2
13D3 0002           CXS2  RMB 2
13D5 0001           SORTF RMB 1
13D6 0002           ENDXS RMB 2
*
* POINT: PROCESS "ENTRY" PSEUDOP
* DEFINES AN ENTRY POINT FOR
* REFERENCE BY OTHER MODULES.
*
13D8 BD 1869 R      POINT JSR  ADPRINT      INIT

```

13DB BD 181A R	JSR LBLCK	CHECK FOR A LABEL
13DE BD 00F6 R	JSR NXTOK	GET ENTRY NAME
13E1 C1 01	CMP B #1	OK?
13E3 27 08	BEQ POENT1	YES
13E5 CE 0216	LDX #S0216	NO, ERROR
13E8 BD 000B R	JSR PRINTE	
13EB 20 39	BRA POENT3	
13ED 7D 0275 R	TST PASS	PASS?
13F0 27 43	BEQ POENT4	PASS 1
13F2 BD 0857 R	JSR LKPSYM	GET ENTRY ADDRESS
13F5 C1 FF	CMP B #SFF	IN SYMTAB?
13F7 26 08	BNE POENT2	YES
13F9 CE 0211	LDX #S0211	NO, ERROR
13FC BD 000B R	JSR PRINTE	
13FF 20 25	BRA POENT3	
1401 FF 0C68 R	STX ADRI	SAVE ENTRY ADDRESS
1404 FE 0282 R	LDX SYMPTR	POINT TO ENTRY
1407 A6 08	LDA A 8,X	GET INFO BYTE
1409 8A 04	ORA A #S04	TURN ON ENT BIT
140B A7 08	STA A 8,X	
140D BD 1438 R	JSR PBLOCK	OUTPUT LABEL
1410 F6 0C68 R	LDA B ADRI	OUTPUT ENTRY ADDRESS
1413 BD 182E R	JSR OUTBIN	
1416 F6 0C69 R	LDA B ADRI	
1419 BD 182E R	JSR OUTBIN	
141C C6 52	LDA B #R	"RELOCATABLE"
141E BD 1842 R	JSR OUTBNR	
1421 C6 4E	LDA B #N	"ENT"
1423 BD 1842 R	JSR OUTBNR	
1426 73 0C66 R	COM POP	
1429 73 027A R	COM ENTFLG	
142C 7C 0C65 R	INC MCOUNT	
142F 7C 0C65 R	INC MCOUNT	
1432 BD 0C0E R	JSR PRINTL	
1435 7E 0490 R	JMP MAIN1	ALL DONE
* * *		
* PBLOCK: ROUTINE TO WRITE LOADER ENTRY SYMBOL ON TAPE		
1438 FE 0282 R	LDX SYMPTR	PT TO ENTRY SYMBOL
143B 86 06	LDA A #6	LENGTH
143D E6 00	LDA B 0,X	GET A CHAR
143F 36	PSH A	
1440 FF 144F R	STX PBXS	
1443 BD 182E R	JSR OUTBIN	
1446 32	PUL A	
1447 FE 144F R	LDX PBXS	
144A 03	INX	
144B 4A	DEC A	ALL DONE?
144C 26 EF	BNE PBLK2	NO
144E 39	RTS	
144F 0002	PBXS RMB 2	
* POEQU: PROCESS EQU PSEUDOP		
1451 BD 1039 R	JSR ADRINT	INIT ADDRESS FIELD FLAGS
1454 FE 0232 R	LDX SYMPTR	
1457 FF 14AD R	STX EQUXS	SAVE SYMPTR
145A 7D 0276 R	1ST LBFLG	LABEL?
145D 26 08	BNE EQU3	YES
145F CE 0213	LDX #S0213	NO, ERROR
1462 BD 000B R	JSR PRINTE	PRINT ERROR
1465 20 40	BRA EQU4	
1467 BD 00F6 R	JSR NXTOK	GET NEXT TOKEN
146A C1 0D	CMP B #S0D	EOL?
146C 26 05	BNE EQU4	NO
146E CE 0216	LDX #S0216	ERROR
1471 20 EF	BRA EQU4	
1473 BD 09B5 R	JSR NSEVL	EVALUATE OPERAND
1476 C1 FF	CMP B #SFF	ERRORS?
1478 27 E8	BEQ EQU4	YES
147A 7D 0275 R	TST PASS	PASS?
147D 26 1F	BNE EQU4	PASS2
147F FE 14AD R	LDX EQUXS	
1482 A6 08	LDA A 8,X	GET INFO BYTE
1484 7D 0277 R	1ST RELFLG	RELOC ?
1487 26 09	BNE EQU4	YES

1489	84 BF	*	AND A #5BF	NO, TURN OFF IN INFO BYTE
148b	7D 0278 R	*	IST CMNFLG	COMMON?
148E	27 02		BEQ EQUF	NO
1490	8A 10	*	ORA A #510	YES TURN ON IN INFO BYTE
1492	A7 08	* EQUF	STA A 8,X	
1494	B6 0C69 R		LDA A ADR2	STORE VALUE
1497	A7 07		STA A 7,X	
1499	B6 0C68 R		LDA A ADR1	
149C	A7 06		STA A 6,X	
149E	73 0C66 R	* EQUQ	COM POP	SET PSEUDOP FLAG
14A1	7C 0C65 R		INC MCOUNT	MCOUNT:=2
14A4	7C 0C65 R		INC MCOUNT	
14A7	BD 0C0E R	* EQUQ	JSR PRINTL	PRINT A LINE OF LISTING
14AA	7E 0490 R		JMP MAIN1	RETURN TO MAIN LOOP
14AD	0002	* EQUXS	RMB 2	SAVE AREA
		*	POEXT1: PROCESS "EXTERNAL" PSEUDOP	
		*	MAKE EXTERNALLY-DEFINED SUBROUTINE	
		*	AVAILABLE TO MODULE	
14AF	BD 1009 R	POEXT1	JSR ADRINT	INIT
14B2	BD 181A R		JSR LBLCK	CHECK FOR A LABEL
14B5	BD 06F6 R	*	JSR NXTOK	GET EXTERNAL ENTRY NAME
14B8	C1 01		CMP B #1	OK?
14BA	27 0B		BEQ POEXT1	YES
14BC	CE 0216	*	LUX #50216	NO, ERROR
14BF	BD 0DBB R		JSR PRINTE	
14C2	BD 0C0E R		JSR PRINTL	
14C5	20 47		BRA POEXT4	
14C7	7C 0234 R	* POEXT1	INC LCN	
14CA	7C 0234 R		INC LCN	
14CD	7C 0234 R		INC LCN	
14D0	7D 0275 R	*	TST PASS	PASS?
14D3	26 0E		BNE POEXT2	PASS 2
14D5	BD 07F8 R	*	JSR STOSYM	PUT NAME IN SYMBOL TABLE
14D8	FE 0232 R		LDX SYMPTR	
14DB	A6 08		LDA A 8,X	
14DD	8A 08		ORA A #508	SET EXT BIT
14DF	A7 08		STA A 8,X	
14E1	20 28		BRA POEXT3	
14E3	C6 7E	* POEXT2	LDA B #57E	"JMP"
14E5	F7 0C67 R		STA B OPCD	
14E8	BD 182E R		JSR OUTBIN	
14EB	BD 0857 R		JSR LKPSYM	SET UP ENTRY NAME
14EE	BD 1438 R		JSR PBLOCK	OUTPUT NAME
14F1	C6 58		LDA B #X	"EXT"
14F3	BD 1842 R		JSR OUTBNR	
14F6	7F 0C68 R		CLR ADR1	
14F9	7F 0C69 R		CLR ADR2	
14FC	7C 0C65 R		INC MCOUNT	
14FF	7C 0C65 R		INC MCOUNT	
1502	7C 0C65 R		INC MCOUNT	
1505	73 0279 R		COM EXTFLG	
1508	BD 0C0E R		JSR PRINTL	
150B	BD 11C2 R	* POEXT3	JSR LCLCN	
150E	7E 0490 R	POEXT4	JMP MAIN1	ALL DONE
		*	POFCB: PROCESS FCB PSEUDOP	
1511	BD 1039 R	POFCB	JSR ADRINT	INIT ADDRESS FIELD FLAGS
1514	BD 06F6 R		JSR NXTOK	GET NEXT TOKEN
1517	C1 0D		CMP B #50D	EOL?
1519	26 08		BNE FCBB	NO
151B	CE 0216	* FCBA	LUX #50216	ERROR
151E	BD 0DBB R		JSR PRINTE	PRINT
1521	20 1A		BRA FCBC	FINISH UP
1523	BD 0985 R	* FCBB	JSR NSEVL	EVALUATE OPERAND
1526	BD 10D7 R		JSR P2ERR	PRINT PASS ERRORS
1529	7C 0234 R		INC LCN	LCN:=1
152C	7D 0275 R		TST PASS	PASS?
152F	27 0F		BEQ FCBD	PASS1
1531	F6 0C69 R	*	LDA B ADR2	PASS2, OUTPUT MC
1534	BD 182E R		JSR OUTBIN	
1537	7C 0C65 R		INC MCOUNT	MCOUNT:=1
153A	7C 0C66 R		INC POP	POP:=1
153D	BD 0C0E R	* FCBC	JSR PRINTL	PRINT A LINE OF LISTING

```

1540 BD 11C2 R FCBJ JSR LCLCN LC*=LC+LCN
1543 7E 0490 R JMP MAIN1 RETURN TO MAIN LOOP
* POFCC: PROCESS FCC PSEUDOP
*
1546 BD 1089 R POFCC JSR ADRINT INIT ADDRESS FIELD FLAGS
1549 BD 06F6 R JSR NXTOK GET NEXT TOKEN
154C C1 27 CMP B #S27 QUOTE ?
154E 27 08 BEQ FCCB YES
*
1550 CE 0204 FCCA LDX #S0204 ERROR
1553 BD 00BB R JSR PRINTE PRINT
1556 20 3C BRA FCCG FINISH UP
*
1558 FE 027E R FCCB LDX CUCHAR GET CURRENT CHAR
155B E6 00 LDA B 0,X
155D C1 0D CMP B #S0D EOL ?
155F 27 EF BEQ FCCA YES
1561 F7 0C69 R STA B ADR2 NO,SAVE CHAR
*
1564 7D 0275 R FCCC TST PASS PASS ?
1567 27 03 BEQ FCCD PASS1
1569 BD 182E R JSR OUTBIN OUTPUT MC
156C FE 027E R FCCD LDX CUCHAR CUCHAR*=CUCHAR+1
156F 08 INX
1570 FF 027E R STX CUCHAR
1573 7C 0284 R INC LCN LCN*=LCN+1
1576 E6 00 LDA B 0,X GET CHAR
1578 C1 27 CMP B #S27 QUOTE?
157A 27 06 BEQ FCCF YES
*
157C C1 0D CMP B #S0D EOL?
157E 26 E4 BNE FCCC NO
1580 20 0C BRA FCCF YES
*
1582 E6 01 FCCF LDA B 1,X GET NEXT CHAR
1584 C1 27 CMP B #S27 TWO QUOTES ?
1586 26 06 BNE FCCF NO
1588 08 INX
1589 FF 027E R STX CUCHAR CUCHAR*=CUCHAR+1
158C 20 D6 BRA FCCC
*
158E 7C 0C66 R FCCF INC * POP POP*=1
1591 7C 0C65 R INC MCOUNT MCOUNT*=1
*
1594 BD 0C0E R FCCG JSR PRINTL PRINT LINE OF LISTING
1597 BD 11C2 R JSR LCLCN LC*=LC+LCN
159A 7E 0490 R JMP MAIN1 RETURN TO MAIN LOOP
* POFDB: PROCESS FDB PSEUDOP
*
159D BD 1089 R POFDB JSR ADRINT INIT ADDRESS FIELD FLAGS
15A0 BD 06F6 R JSR NXTOK GET NEXT TOKEN
15A3 C1 0D CMP B #S0D EOL?
15A5 26 08 BNE FDBB NO
*
15A7 CE 0216 FDBA LDX #S0216 ERROR
15AA BD 00BB R JSR PRINTE PRINT
15AD 20 39 BRA FDBC FINISH UP
*
15AF BD 09B5 R FDBB JSR NSEVL EVALUATE OPERAND
15B2 BD 10D7 R JSR P2ERR PRINT PASS 2 ERRORS
*
15B5 7C 0284 R INC LCN LCN*=2
15B8 7C 0284 R INC LCN
15BB 7D 0275 R TST PASS PASS?
15BE 27 2B BEQ FDBD PASS1
*
15C0 F6 0C68 R LDA B ADR1 OUTPUT MC
15C3 BD 182E R JSR OUTBIN
15C6 F6 0C69 R LDA B ADR2
15C9 BD 182E R JSR OUTBIN
15CC 7D 0277 R TST RELFLG RELOC ?
15CF 27 04 BEQ FDCF NO
*
15D1 C6 52 LDA B #'R YES
15D3 2D 07 BRA FDCG
*
15D5 7D 0278 R FDCF TST CMNFLG COMMON?
15D8 27 05 BEQ FDCE NO
*
15DA C6 4D LDA B #'M YES
*
15DC BD 1842 R FDCG JSR OUTBNR OUTPUT "R" OR "M"
15DF 7C 0C65 R FDCE INC MCOUNT MCOUNT*=2
15E2 7C 0C65 R INC MCOUNT
15E5 73 0C66 R COM POP POP*=FF
*
15E8 BD 0C0E R FDBC JSR PRINTL PRINT A LINE OF LISTING
15EB BD 11C2 R FDBD JSR LCLCN LC*=LC+LCN
15EE 7E 0490 R JMP MAIN1 RETURN TO MAIN LOOP
* PROCESS THE IF PSEUDOP
*

```

```

15F1 BD 1089 R POIF JSR ADRINT
15F4 BD 181A R JSR LBLCK
15F7 BD 06F6 R JSR NXTOK
15FA C1 0D CMP B #$0D EOL?
15FC 26 08 BNE POIFB NO

*
15FE CE 0216 POIFA LDX #$0216
1601 BD 00BB R JSR PRINTE
1604 20 23 BRA POIFE

*
1606 BD 09B5 R POIFB JSR NSEVL EVALUATE OPERAND
1609 C1 FF CMP B #$FF ERRORS?
160B 27 F1 BEQ POIFA YES

*
160D BD 176A R JSR PSHIF STACK PRESENT IFFLG
1610 7D 0377 R TST IFFLG ASSEMBLING?
1613 27 14 BEQ POIFE NO

*
1615 7D 0C68 R TST ADR1 =0?
1618 26 0A BNE POIFC NO

*
161A 7D 0C69 R TST ADR2 =0?
161D 26 05 BNE POIFC NO

*
161F 7F 0377 R CLR IFFLG TURN OFF ASSEMBLING
1622 20 05 BRA POIFE

*
1624 86 FF POIFC LDA A #$FF TURN ON ASSEMBLING
1626 B7 0377 R STA A IFFLG

*
1629 BD 0C0E R POIFE JSR PRINTL
162C 7E 0490 R JMP MAINI

* PROCESS THE MAC PSEUDOP
*
162F 0001 CMFLG RMB 1 COMMENT FLAG 0=NO,FF=YES
1630 0001 MACERR RMB 1 MAC ERROR 0=NO,FF=YES

*
1631 BD 1089 R POMAC JSR ADRINT INIT FLAGS
1634 BD 0C0E R JSR PRINTL
1637 7F 162F R CLR CMFLG
163A 7F 1630 R CLR MACERR
163D 7D 0275 R TST PASS PASS?
1640 26 30 BNE POMAC2 PASS2

*
1642 7D 0276 R TST LBFLG LABELED?
1645 26 0B BNE POMAC1 YES,OK

*
1647 73 1630 R COM MACERR SET ERROR FLAG
164A CE 0226 LDX #$0226 ERROR
164D BD 00BB R JSR PRINTE
1650 20 20 BRA POMAC2

*
1652 FE 0282 R POMAC1 LDX SYMPTR PT TO LABEL
1655 86 20 LDA A #$20
1657 A7 08 STA A 8,X SET MACRO FLAG IN SYMTAB
1659 B6 030D R LDA A MACPTR
165C A7 06 STA A 6,X SAVE MACRO LOC
165E B6 030E R LDA A MACPTR+1
1661 A7 07 STA A 7,X

*
1663 BD 06F6 R JSR NXTOK CHECK FOR "C"
1666 FE 027B R LDX DESCRA
1669 A6 00 LDA A 0,X
166B 81 43 CMP A #'C "C"?
166D 26 03 BNE **5 NO

*
166F 73 162F R COM CMFLG YES,SAVE COMMENTS

*
1672 BD 0576 R POMAC2 JSR RDLINA GET NEXT LINE
1675 FE 026F R LDX LNUM
1678 08 INX
1679 FF 026F R STX LNUM
167C BD 0C0E R JSR PRINTL
167F FE 0280 R LDX CULINE PT TO LINE
1682 A6 00 LDA A 0,X GET FIRST CHAR
1684 81 2A CMP A #'* COMMENT?
1686 26 0A BNE POMAC5 NO

*
1688 7D 162F R TST CMFLG SAVE COMMENTS?
168B 27 E5 BEQ POMAC2 NO

*
168D BD 16E6 R JSR MACMOV YES, SAVE

*
1690 20 E0 BRA POMAC2

*
1692 7F 0276 R POMAC5 CLR LBFLG CLEAR LABEL FLAG
1695 FE 0280 R LDX CULINE PT TO LINE
1698 A6 00 LDA A 0,X GET CHAR
169A 81 20 CMP A #$20 BLANK?
169C 27 06 BEQ POMAC6 YES

*
169E BD 06F6 R JSR NXTOK GET LABEL

```

16A1	73	0276	R		COM	LBFLG	SET LABEL FLAG
*							
16A4	8D	06F6	R	POMAC6	JSR	NXTOK	GET MNEMONIC
16A7	86	04			LDA	A #4	SET FOR COMPARE
16A9	87	07E7	R		STA	A PCOUNT	
16AC	FE	027B	R		LDX	DESCRA	
16AF	FF	07E3	R		STX	PSING1	
16B2	CE	1722	R		LDX	#MEND	
16B5	FF	07E5	R		STX	PSING2	
16B8	CE	07E3	R		LDX	#PSING1	POINT TO PARMS
16Bb	BD	06C5	R		JSR	COMPAR	COMPARE
16BE	26	0D			BNE	POMAC8	MEND NOT FOUND
*							
16C0	7D	0276	R		TST	LBFLG	LABELED?
16C3	27	0E			BEQ	POMAC7	YES
*							
16C5	CE	0227			LDX	#0227	ERROR
16C8	BD	0DBB	R		JSR	PRINTE	
16CB	2D	06			BRA	POMAC7	
*							
16CD	BD	16E6	R	POMAC8	JSR	MACMOV	PUT INTO MACTBL
16D0	7E	1672	R		JMP	POMAC2	
*							
16D3	7D	0275	R	POMAC7	TST	PASS	PASS?
16D6	26	0B			BNE	POMACA	PASS2
*							
16D8	86	17			LDA	A #517	ETB TO END OF MACRO
16DA	FE	030D	R		LDX	MACPTR	
16DD	A7	00			STA	A 0,X	
16DF	08				INX		
16E0	FF	030D	R		STX	MACPTR	
*							
16E3	7E	0490	R	POMACA	JMP	MAIN1	ALL DONE
*							
* MOVE MACRO TO MACTBL							
*							
16E6	7D	0275	R	MACMOV	TST	PASS	PASS?
16E9	26	36			BNE	MACMVE	PASS2
*							
16EB	7D	1630	R		TST	MACERR	ERROR?
16EE	26	31			BNE	MACMVE	YES
*							
16F0	FE	0280	R		LDX	CULINE	
16F3	FF	027E	R		STX	CUCHAR	
*							
16F6	FE	027E	R	MACLOP	LDX	CUCHAR	GET CHAR FROM INBUF
16F9	A6	00			LDA	A 0,X	
16FB	08				INX		
16FC	FF	027E	R		STX	CUCHAR	
16FF	FE	030D	R		LDX	MACPTR	POINT TO MACTBL
1702	BC	0264	R		CPX	MACEND	FULL?
1705	26	10			BNE	MACMVI	NO
*							
1707	86	0D			LDA	A #50D	CR TO MACTBL
1709	A7	00			STA	A 0,X	
170B	08				INX		
170C	FF	030D	R		STX	MACPTR	
*							
170F	CE	0228			LDX	#0228	ERROR
1712	BD	0DBB	R		JSR	PRINTE	
1715	20	0A			BRA	MACMVE	
*							
1717	A7	00		MACMVI	STA	A 0,X	STORE CHAR IN MACTBL
1719	08				INX		
171A	FF	030D	R		STX	MACPTR	
171D	81	0D			CMP	A #50D	EOL?
171F	26	D5			BNE	MACLOP	NO
*							
1721	39			MACMVE	RTS		ALL DONE
*							
1722	4D			MEND	FCC	'MEND'	
* PONAM* PROCESS NAM PSEUDOP							
*							
1726	BD	1089	R	PONAM	JSR	ADRINT	
1729	BD	181A	R		JSR	LBLOCK	
*							
172C	BD	06F6	R		JSR	NXTOK	GET PROGRAM NAME
172F	C1	01			CMP	B #1	OK ?
1731	27	09			BEQ	PONAM1	YES
*							
1733	CE	0216			LDX	#0216	ERROR
1736	BD	0DBB	R		JSR	PRINTE	
1739	7E	1426	R		JMP	POENT3	
*							
173C	7D	0275	R	PONAM1	TST	PASS	PASS?
173F	26	06			BNE	PONAM2	PASS 2
*							
1741	BD	07F8	R		JSR	STOSYM	SAVE NAME IN SYMTAB
1744	7E	13ED	R		JMP	POENT1	
*							
1747	F6	0313	R	PONAM2	LDA	B CMNLC	OUTPUT COMMON BLOCK SIZE
174A	BD	182E	R		JSR	OUTBIN	

```

174D F6 0314 R      LDA B CMNLC+1
1750 BD 182E R      JSR OUTBIN

*
1753 C6 50          LDA B #P          "PROGRAM"
1755 BD 1842 R      JSR OUTBNR

*
1758 7E 13ED R      JMP POENTI        PROCESS AS ENTRY NAME
* PONIF: PROCESS NIF PSEUDOP
*
175B BD 1089 R      PONIF JSR ADRINT
175E BD 181A R      JSR LBLCK
1761 BD 177E R      JSR PULIF        GET LAST IFFLG
1764 BD 0C0E R      JSR PRINTL
1767 7E 0490 R      JMP MAINI

*
*
* PSHIF: PUSH THE CURRENT IFFLG ONTO THE IFSTACK
*
176A BF 028C R      PSHIF STS STKSAV    SAVE STACK POINTER
176D FE 0375 R      LDX #IFSTK        LOAD IF STACK POINTER
1770 8C 036D R      CPX #IFSTK-8      FULL?
1773 27 1D          BEQ PSPLER        YES

*
1775 BE 0375 R      LDS #IFSTK        LOAD STACK POINTER
1778 B6 0377 R      LDA A IFFLG
177B 36             PSH A             STACK IFFLG
177C 20 1B          BRA PSPLCM

*
* PULIF: PULL LAST IFFLG OFF OF THE IFSTACK
*
177E BF 028C R      PULIF STS STKSAV    SAVE STACK POINTER
1781 FE 0375 R      LDX #IFSTK        LOAD IF STACK POINTER
1784 8C 0375 R      CPX #IFSTK        UNDERFLOW?
1787 27 09          BEQ PSPLER        YES

*
1789 BE 0375 R      LDS #IFSTK        LOAD STACK POINTER
178C 32             PUL A             POP LAST IFFLG
178D B7 0377 R      STA A IFFLG
1790 20 07          BRA PSPLCM

*
1792 CE 0254 R      PSPLER LDX #S0254
1795 BD 0DBB R      JSR PRINTE
1798 39             RTS

*
1799 BF 0375 R      PSPLCM STS #IFSTK
179C BE 028C R      LDS STKSAV
179F 39             RTS

* POPAG: PROCESS PAG PSEUDOP
*
17A0 BD 1089 R      POPAG JSR ADRINT    INIT FLAGS
17A3 BD 181A R      JSR LBLCK          CHECK FOR LABEL
17A6 7D 0275 R      TST PASS          PASS ?
17A9 27 13          BEQ PAGEND        PASS 1

*
17Ab F6 0287 R      LDA B LCOUNT     LCOUNT=0?
17AE 27 0E          BEQ PAGEND        YES

*
17B0 C6 3C          LDA B #S3C        B:=60
17B2 F0 0287 R      SUB B LCOUNT     B:=60-LCOUNT

*
17B5 BD 13BE R      PAGEA JSR CRLF
17B8 5A             DEC B
17B9 26 FA          BNE PAGEA         TO
17BB 7F 0287 R      CLR LCOUNT       TOP OF PAGE
17BE 7E 0490 R      PAGEND JMP MAINI   LCOUNT:=0
* PORMB: PROCESS RMB PSEUDOP
*
17C1 BD 1089 R      PORMB JSR ADRINT    INIT ADDRESS FIELD FLAGS
17C4 BD 09F6 R      JSR NX10K         GET NEXT TOKEN
17C7 C1 0D          CMP B #S0D        EOL?
17C9 26 08          BNE RMBB          NO

*
17CB CE 0216 R      LDX #S0216        ERROR
17CE BD 0DBB R      RMBB JSR PRINTE    PRINT
17D1 20 18          BRA RMBB          FINISH UP

*
17D3 BD 09B5 R      RABB JSR NSEVL     EVALUATE OPERAND
17D6 C1 FF          CMP B #SFF        ERRORS?
17D8 27 F4          BEQ RMBB          YES

*
17DA 7D 0275 R      TST PASS          PASS?
17DD 27 0F          BEQ RMBB          PASS1

*
17DF BD 1803 R      JSR RMBOUT        OUTPUT MC
17E2 7C 0C65 R      INC MCOUNT        MCOUNT:=2
17E5 7C 0C65 R      INC MCOUNT
17E8 73 0C66 R      COM POP          SET PSEUDOP FLAG

*
17EB BD 0C0E R      RMBB JSR PRINTL    PRINT A LINE OF LISTING
17EE B6 0274 R      LDA A LC+1        LC:=LC+ADR1,ADR2
17F1 F6 0273 R      LDA B LC
17F4 BB 0C69 R      ADD A ADR2

```

```

17F7 F9 0C68 R      ADC B ADRI
17FA B7 0274 R      STA A LC+1
17FD F7 0273 R      STA B LC
1800 7E 0490 R      JMP MAINI      RETURN TO MAIN LOOP
*
1803 5F             *
1804 FE 0C68 R      RMBOUT CLR B      LOAD 00
1807 FF 0285 R      LDX ADRI      LSAVE:=#BYTES FROM RMB
1807 FF 0285 R      STX LSAVE
*
180A BD 182E R      RMBOTA JSR OUTBIN  OUTPUT MC
180D FE 0285 R      LDX LSAVE
1810 09             DEX
1811 27 06             BEQ RMBOTB  DONE
*
1813 FF 0285 R      STX LSAVE
1816 5F             CLR B
1817 20 F1             BRA RMBOTA  DO AGAIN
*
1819 39             RMBOTB RTS      RETURN
* LBLCK: CHECK FOR A AN ILLEGAL LABEL ON A
* PSEUDOP. IF THERE IS ONE DELETE IT,
* AND PRINT AN ERROR MESSAGE.
*
181A 7D 0276 R      LBLCK 1ST LBFLG  LABEL?
181D 27 0E             BEQ LBLCK2  NO
*
181F 7D 0275 R      1ST PASS?
1822 26 03             BNE LBLCK1  PASS2
*
1824 BD 08AB R      JSR DELSYM  PASS1 DELETE LAST SYMBOL
*
1827 CE 0223 R      LBLCK1 LDX #0223  ERROR
182A BD 00BB R      JSR PRINTE  PRINT
*
182D 39             LBLCK2 RTS      RETURN
* OUTBIN: OUTPUT A BYTE AS TWO HEX ASCII CHARACTERS
* OUTBNR: OUTPUT "R", "N", OR "X"
*
182E B6 026E R      OUTBIN LDA A OPTNS
1831 85 40             BIT A #540
1833 26 0C             BNE OUTRET  OUTPUT?
*
1835 17             TBA
1836 BD 16             BSR OUTHL  CONVERT LEFT NIBBLE
1836 BD 0256 R      JSR OUTB
183B 17             TBA
183C BD 14             BSR OUTHR  CONVERT RIGHT NIBBLE
183E BD 0256 R      JSR OUTB
1841 39             OUTRET RTS
*
1842 B6 026E R      OUTBNR LDA A OPTNS
1845 85 40             BIT A #540
1847 26 F8             BNE OUTRET  OUTPUT?
*
1849 17             TBA
184A BD 0256 R      JSR OUTB
184D 39             RTS
*
184E 44             OUTHL LSR A

184F 44             LSR A
1850 44             LSR A
1851 44             LSR A
1852 84 0F           OUTHR AND A #50F
1854 8B 30           ADD A #530
1856 81 39           CMP A #539
1858 23 02           BLS **4
185A 8B 07           ADD A #507
185C 39             RTS
* ASSORTED I/O ROUTINES
*
185D BD 1888 R      PDATA2 JSR OUTCHR
1860 08             INX
1861 A0 00           PDATA1 LDA A 0,X
1863 81 04           CMP A #4
1865 26 F6           BNE PDATA2
1867 39             RTS
*
1868 A6 00           OUT2H LDA A 0,X
186A 8D 0E           OUT2HA BSR OUTHLL
186C A6 00           LDA A 0,X
186E 08             INX
186F 2D 0J           BRA OUTHR
*
1871 8D F5           OUT4HS BSR OUT2H
1873 8D F3           OUT2HS BSR OUT2H
1875 86 20           LDA A #520
1877 7E 1888 R      OUTS JMP OUTCHR
*
187A 44             OUTHLL LSR A
187B 44             LSR A
187C 44             LSR A
187D 44             LSR A
*
187E 84 0F           OUTHR AND A #50F
1880 8B 30           ADD A #530
1882 81 39           CMP A #539
1884 23 02           BLS OUTCHR
*
1886 8B 07           ADD A #507
*
1888 36             OUTCHR PSH A
1889 BD 038B R      JSR OUTEEE
188C 32             PUL A
188D 81 0A           CMP A #50A  LF?
188F 26 0E           BNE OUTCHE  NO
*
1891 36             PSH A
1892 37             PSH B
1893 C6 08           LDA B #8
*
1895 86 00           OUTCHL LDA A #500
1897 BD 038B R      JSR OUTEEE
189A 5A             DEC B
189B 26 F8           BNE OUTCHL
*
189D 33             PUL B
189E 32             PUL A
189F 39             OUTCHE RTS
*
END

```

#IFSJK	0375 R	CVHBI	0A0B R	LCN	0234 R	NSVLB	09E4 R
ABR	1084 R	CVHBD	0B14 R	LCN2	1131 R	NSVLC	09EC R
ABRCK	10B7 R	CVHBS	0B18 R	LCN2A	1145 R	NSVLC1	09F2 R
ABRCKA	10C0 R	CXS2	13D3 R	LCN2B	1157 R	NSVLD	09F7 R
ADDR16	0BEC R	DCOUNT	0B25 R	LCN3	117C R	NSVLE	09FA R
ADDR1A	0E00 R	DEL1	08B2 R	LCN3A	118A R	NSVLF	09FD R
ADDR1A	0E0A R	DELSYM	08AB R	LCN3B	1186 R	NSVLG	0A07 R
ADDR1B	0E12 R	DESCRA	027B R	LCN3C	11A9 R	NSVLH	0A15 R
ADDR1C	0E35 R	DESCRC	027D R	LCNAB1	10EA R	NSVLH1	0A25 R
ADDR1D	0E4C R	DIV151	0BB1 R	LCNAB2	1113 R	NSVLJ	0A2A R
ADDR1E	0E6D R	DIV153	0BB3 R	LCNAB3	115E R	NSVLJ1	0A3A R
ADDR1F	0E72 R	DIV16	0BA1 R	LCOUNT	0287 R	NSVLK	0A3F R
ADDR1G	0E7E R	DIV163	0BC6 R	LINCK	0C2A R	NSVLL	0A46 R
ADDR1H	0E88 R	DIV165	0BD3 R	LINCKA	0C35 R	NSVLLA	0A5F R
ADDR1J	0E8B R	DIV167	0BD4 R	LINCKB	0C42 R	NSVLM	0A65 R
ADDR1K	0E40 R	DSCAN	075C R	LINEN	0C6A R	NSVLN	0A73 R
ADDR1L	0E63 R	DVAL	0B23 R	LKPSM1	085D R	NSVLP	0A7C R
ADDR2	0E91 R	DJSAV	0B26 R	LKPSM2	0863 R	NSVLP1	0A8B R
ADDR2A	0E9B R	ECOUNT	0288 R	LKPSM3	0866 R	NSVLP2	0A97 R
ADDR2B	0EA3 R	ENDMA	1395 R	LKPSM4	0873 R	NSVLP3	0AB0 R
ADDR2E	0EJ5 R	ENDMB	13A1 R	LKPSYM	0857 R	NSVLP4	0AB7 R
ADDR2F	0EJA R	ENDP2	1276 R	LNAB1B	1100 R	NSYM	026A R
ADDR2G	0EE6 R	ENDP3	1283 R	LNAB1C	1103 R	NXT0	06FC R
ADDR2H	0EF0 R	ENDP4	12F3 R	LNAB10	1106 R	NXT1	0714 R
ADDR2J	0EF3 R	ENDP5	1340 R	LNAB1S	110F R	NXT3	071A R
ADDR2K	0ECB R	ENDP6	134D R	LNAB2S	112C R	NXT3A	072E R
ADDR3	0EF9 R	ENDP6A	1381 R	LNAB3S	1177 R	NXT4	0734 R
ADDR3B	0F0B R	ENDP7	1367 R	LN82	1129 R	NXT4A	0736 R
ADDR3C	0F22 R	ENDUSCH	07B6 R	LN83	1174 R	NXT5	073C R
ADDR3D	0F35 R	ENDXS	1306 R	LN8M	026F R	NXT6	0744 R
ADDR3E	0F3A R	ENSIZ	093E R	LP	093B R	NXTER	0759 R
ADDR3F	0F42 R	ENTFLG	027A R	LSAVE	0285 R	NXTOK	06F6 R
ADDR4	0F48 R	EQUA	1462 R	MACEND	0264 R	OPCJ	0C67 R
ADDR4A	0F5C R	EQUB	1467 R	MACERR	1630 R	OPERN	09B4 R
ADDR5	0F68 R	EQUC	1473 R	MACFLG	030C R	OPIN	0391 R
ADDR5A	0F72 R	EQUJ	149E R	MACLIN	0292 R	OPIN1	039D R
ADDR5B	0F7A R	EQUE	14A7 R	MACLOP	16F6 R	OPIN2	03C2 R
ADDR5C	0F97 R	EQUF	1492 R	MACMOV	16E6 R	OPIN3	03CA R
ADDR5D	0FA4 R	EQUXS	14AD R	MACMVI	1717 R	OPTMSG	0378 R
ADDR5E	0FC0 R	ERASA	0DA3 R	MACMVE	1721 R	OPTNS	026E R
ADDR5F	0FC5 R	ERMSB	0DAF R	MACPAR	020A R	ORBYA	1037 R
ADDR5G	0FCC R	ERMSC	0DB3 R	MACPSH	0637 R	ORBYB	1088 R
ADDR5H	0FJ1 R	ERNUM	0DA1 R	MACPTR	030D R	OUT2H	1068 R
ADDR5J	0FJA R	EXTFLG	0279 R	MACPUL	069B R	OUT2HA	186A R
ADDR5K	0F8B R	FC6A	151B R	MACSAV	030F R	OUT2HS	1873 R
ADDR6	0FJA R	FC6B	1523 R	MACSTK	0266 R	OUT4HS	1871 R
ADDR7	0FE6 R	FCBC	153D R	MACIBL	0262 R	OUTB	0256 RX
ADDR7A	0FF8 R	FCBJ	1540 R	MAIN1	0490 R	OUTBIN	182E R
ADDR7B	1006 R	FCCA	1550 R	MAIN10	0544 R	OUTBNR	1842 R
ADDR7C	1010 R	FCCB	1558 R	MAIN11	0550 R	OUTCHE	189F R
ADDR7D	1013 R	FCCD	1564 R	MAIN12	054A R	OUTCHL	1895 R
ADDR8	1019 R	FCCD	156C R	MAIN13	0547 R	OUTCHR	1888 R
ADDR8A	102B R	FCCE	1582 R	MAIN1A	04A6 R	OUTEEE	033B R
ADDR8C	105E R	FCCF	158E R	MAIN3	04AE R	OUTHL	184E R
ADDR8D	1061 R	FCCG	1594 R	MAIN3A	04BA R	OUTHLL	187A R
ADDR8E	1051 R	FDBA	15A7 R	MAIN3B	04D3 R	OUTHR	1852 R
ADDR8F	1058 R	FDBB	15AF R	MAIN3C	04D5 R	OUTHRR	187E R
ADDR9	106A R	FDBC	15E6 R	MAIN4	04EB R	OUTL	0C71 R
ADDR9A	1075 R	FBDJ	15EB R	MAIN5	04F6 R	OUTL1	0CC2 R
ADR1	0C66 R	FJCE	15D7 R	MAIN6	04FD R	OUTL2	0CD0 R
ADR2	0C69 R	FJCF	15D5 R	MAIN7	0506 R	OUTL3	0CD0 R
ADRINT	1089 R	FJCG	15DC R	MAIN7A	0527 R	OUTL4	0CE1 R
ASM	0000 RN	GCHRTB	07C3 R	MAIN8	053E R	OUTL5	0D01 R
BLANK	0D53 R	GCHTR	07DD R	MAIN9	053C R	OUTL6	0D07 R
BLANK3	0D50 R	GETB	0253 RX	MCLPTR	0311 R	OUTL6A	0D2B R
BLANK5	0D58 R	HASH	0889 R	MCOUNT	0C65 R	OUTL6B	0D2D R
BLANK6	0D59 R	HASH1	08D4 R	MCRLF	13C5 R	OUTL7	0D38 R
CBLOCK	13D1 R	HEADR	0C4B R	MEND	1722 R	OUTL7A	0D35 R
CDONE	06F1 R	HKEYA	07F0 R	MHLI	099E R	OUTLA	0CA5 R
CHPTR	070F R	HKEYB	07F2 R	MNLKP	0940 R	OUTREI	1841 R
CHRTAB	020A R	HSAY1	07F4 R	MNLKPA	094F R	OUTS	1875 R
CLASS	09B3 R	HSAY2	07F6 R	MNLKPB	095B R	P2ERR	10J7 R
CLFLG	0982 R	HSCAN	0798 R	ANMI	09A6 R	P2ERRA	10E3 R
CAFLG	162F R	HSCAN1	07A1 R	MNTAB	0006 R	P2ERRB	10E9 R
CMNFLG	0278 R	HSMBL	07EA R	MONIOR	0250 RX	PAGEA	17B5 R
CMNLC	0313 R	HVAL	0ACC R	MP	093C R	PAGEND	17BE R
CMNXS	124C R	IFFLG	0377 R	MPSH1	0645 R	PASS	0275 R
CMP1	06CC R	IFSJK	0375 R	MPSH2	065C R	PASS1	038E R
CMP2	06DC R	IMMED	1085 R	MPSH3	0665 R	PASS2	0407 R
CMP3	06EE R	INDEX	1036 R	MPSH5	0685 R	PBLK2	143D R
CMSG	13AE R	INEEE	0388 RN	MPUL1	06A7 R	PBLOCK	1438 R
COMPAR	06C5 R	INITIO	025C RX	MPUL2	06B4 R	PBX5	144F R
CRLF	13BE RN	INLINE	0315 R	MPY16	0B7D R	PCOUNT	07E7 R
CUCAR	027E R	INXCK	10C4 R	MPY163	0B8B R	PDATA1	1861 RN
CULINE	0280 R	INXCKR	10D3 R	MPY167	0B97 R	PDATA2	185D R
CVBJD	0J5C R	IP	093D R	MSKPT	023A R	PLEND	0C29 R
CVDB	0B2A R	KIOK	0D92 R	MXSAV1	028E R	POCMN	11D4 R
CVDB1	0B43 R	LBFLG	0276 R	MXSAV2	0290 R	POCMN0	11E1 R
CVDB2	0B47 R	LBLCK	181A R	NSCAN	0773 R	POCMN1	11E4 R
CVDEC1	0D62 R	LBLCK1	1827 R	NSCANA	0794 R	POCMN2	11E9 R
CVDEC2	0D65 R	LBLCK2	182D R	NSEVL	09B5 R	POCMN3	122F R
CVDEC5	0D70 R	LC	0273 R	NSVLA	09D3 R	POCMN4	1246 R
CVHB	0ACE R	LCLCN	17C2 R				

```

S1137040bEA0427E03CE41424110AA1B4144430E7E
S113705400U941444400430414E440E4904415373
S113706A0C4UF39084153520F390742434310592466
S1137070424353105925424551105927427451000
S1137090592Q4247410592E42484910592242498A
S11370930540E4905424C45105927424C5310592330
S11300A0424C5410592J42494U10592B424E451083
S1137080592642504C10592A425241105920425395
S11370UC05210598U425643105928425653105929F8
S11300U043424110A111434C410AA0C434C49100B
S11300E0AA0E434C510A0F350F434C5610AA0A34DE3
S11370F0461214F434B050E4001434F40DF3F0330
S11370104350580FA884C441110AA1944454304340
S1130110300A44455310AA3444455610AA09454E97
S113012044128E4F454B54148FF454F20E4A0089A
S11301304551551491FF45J85414E2CFF464342155C
S11301404EEF444343158FF44444215DAFF4A1682
S113015020162E4F444E4430F390C44E510A310A313
S1130160494E510AA004A4AD5010266E4A53521050
S113017026A4A4410E4000A444530FA8E4C44CB
S1130180580FA8CE4453520F390444D4143166EFFFDD
S11301904E414U1763FF4E45470F3900E4494617F0
S11301A098F4E4F5010AA024F32410E4005A01403E
S11301B0471700FF050348CF883605054C0F8832FF6
S11301C0524D4217FEF5F5244C0F3909524F520FF6
S11301D0390652544910AA3B52545310AA39534277
S11301E04110AA10534224C0400253454310A00D368
S11301F053454910AA0F35455610AA085354410E3A8
S1130200D107535453101A8F535458101ACF5355B8F
S1130210420E400053574910AA3F5441210AA1A6B7
S11302205445010AA0654424110AA175450411098
S1130230AA075453540F390D54535810AA30545824
S11302403410AA3557414910AA3E0000000404843

```

[illegible]

S1130JDF000010200000000000203A043637FF0DE117
S1130E00B600UE18B30870UEFBF00DE2444444438BC
S1130E1030B70DF0FB00DE2840F830B70DF1CE0D67
S1130E20F3B602AFF6020B0D00F5CCE0DE3BD18E925
S1130E30B_J0U75332FE02C808FFD2C8FE0DE1394C
S1130E40B010C9BD0736C10D2608CE0204BD0DFB79
S1130E502058B310DF7F510C427F0BD0736C123266A
S1130E60147710C5B0D0736C12/260AFC0E2BEA600AC
S1130E70B70CA9200B0D9F5B01117F610C5270C39
S1130E80C680F710C76C0FF10C8203BD0736BD2
S1130E901104262A7D02B8260A7D02B72605F60C1F
S1130EA0A8270FC6B0F710C7C6F0F710C8BD119E2B
S1130EB02019C690F710C7C6D0F710C8200AC6A0DC
S1130EC0F710C7C6E0F710C8BD1153BD12027E0467
S1130EED0B0D10C9BD0736C10U2608CE0204B0D014
S1130EE0FB2032BD10F7F610C427F0BD0736BD094C
S1130EF0F5BD1117BD0736BD1104262A7D02B82694C
S1130F00A47D02B72605F60CA8270FC6B0F710C74E
S1130F10C6F0F710C8BD119E2019C690F710C7C6B9
S1130F20D0F71J0C8200AC6A0F10C7C6E0F710C84B
S1130F30BD1153BD12027E04D0BDF10C9BD0736C118
S1130F40D2608CE0204BD0DFB202ABD10F77D10E2
S1130F50C4270FC640F710C7C650F710C8BD112AE2
S1130F602020BD0F9C5BD1117BD0736BD1104260AA1
S1130F70C670F710C7BD11BC2008C660F710C7BD06
S1130F801171BD12027E04D0BD10C9BD0736BD105B
S1130F90F77D10C42606CE0204B0D0F710C8BD2F
S1130FA0112ABD12027E04D0BD10C9BD0736C10D81
S1130FB02608CE0204BD0DFB2046C123261973101E
S1130FC0C5BD0736C127260F022BEA600B0CA872
S1130FJA0A61870CA92029BD0F95BD1117F610C546
S1130FE02702201CBD0736BD110426207D02B82629
S1130FF0A7D02B72605F60CA8270AC630F710C7E3
S1131000B011BC200FC610F710C72005C620F7106D
S1131010C7BD1171BD12027E04D0BD10C9BD073631
S1131020C10D2683208CBD10C9BD0736C10D2608DD
S1131030CE0204BD0F9B200EBD09F5BD1117BD0781
S113104306BD1104260AC610F710C7BD11BC200313
S1131050BD1117BD12027E04D0BD10C9BD0736C19D
S11310600D2608CE0204BD0DFB20367D02B52731C6
S1131070BD09F5BD1117FE023J0808FFD20C5B60C81
S1131080A9F60CA8B002C6F20C5C1FF26032D2877
S11310900UC10026034J2A06CE0208B0D0FRB70C78
S113110A0A9BD1171BD12027E04D0BD10C97D02B567
S113110B02703BD18687C0CA57C02C4B00CA4EBD126D
S113110C0027E04D000000000007F02C47F02B770C77
S113110D002887F02B97F02BA7F0CA67F0CA77F0CC
S113110E0A57F10C47F10C57F10C67F0CA87F0CA97
S113110F07F10C77F10C839C1412705CA14227013974
S1131100F710C439C12C2608BD0736C15826047309
S113111010C6397F10C639C1FF2605F7D02B52703DC
S11311120B0J0F7F0CA87F0CA87F0CA87F0CA87F0CA87
S113111300CA7B610C4270F81422705FA10C7200355
S11311140FA10C3F70CA7BD18687F0CA5B0C4E7C1F
S113111502C4397D02B5273FF60CA7B610C4272573
S1131116081422705FA10C72003FA10C8F70CA720FC
S11311170147D02B52721F702B97F02B870CA7FAC7
S1131118010C7F70CA7BD1868F0CA9BD18687C0C27
S11311190A57F0CA5B0C4E7C02C47C02C4397U0226
S113111A0B52755F60CA7B610C4271F81422705FA8
S113111B010C72003FA10C8F70CA7200E7D02B5272C
S113111C037F0CA7FA10C7F70CA7BD1868F0CA8B6
S113111D0B01868F0CA9BD18687D02B82704CA64BD
S113111E02007J02B72705C652BD187F70CA57C5U
S113111FO0CA57C0CA5B0C4E7C02C47C02C47C02F4
S1131200C439B602B4F602B3B0C2AC90B702B40F
S1131210F702B339BD10C9BD1857BD0736C1012740
S113122008CE0216B0D0F8205D7D02B52641BD082A
S113123038FE02C2FF128C8D0736C12C26E3BD075F
S113124036BD09F5C1FF27DCFE12dC868FA4088ACF
S113125010A708B60353A706B6035A4707B60CA9EC
S1131260F60CA8B0354F90353B70354F70353BD57
S113127008F7FE02CE2EE06FF0CA8730CA67302B810
S11312807C0CA57C0CA5B0C4E7E04D00000BD10CA
S1131290C9BD1J577D02B5260FFE02B3FF02B17314
S11312A002B5BD029F7E04A7FE02B81BC02B370A6AD
S11312B0C0E020BD0DFB602AE35802606B0C4EC7
S11312C0B013FE6B02AE652027037E1380C41080F
S11312D0FF14117F1415FE02A32007980E08080845
S11312E008080J08BC02AC26087D141527037E13DE
S11312F02B7E138DE600C12027E11E608C1FF27DB2
S1131300FF1413FF0825FE1411FF0823C606F7086F
S113131027C0E432BD07052205FE141320BDFF14A5
S113132013FF14116FF5FF7141520B0806AC60A6CE
S1131330FE1411A600BD18C5085A26F78620BD184C
S1131340C5BD18AEFF1416E600C4052705852BUC7
S113135013C5C520705864JBD18C5C510270586A7
S1131360A3BD18C5C50827058658BD18C5C504273B
S113137005B64EB18C5E6002A06CE13CABD189EC2
S1131380FE1410C6FFE700BD13FE7E12CDBD13FE8C
S1131390BD13FECE13E18602C8F602C9BD0D9CCE44
S11313A013D5BD189EBD13FEBD13FECE13EEBD189E
S11313B09CE0C53BD18AE6D13FEB602AE85A026C5
S11313C006HJ02YU7E02BD7E02902524445461

S11313D0494E454404544845524520574552453AE0
S11313E020000000000204552524F525304434F46
S11313F04D4D4F4E204C454E4754483D2004CE148D
S113140005BD189E390D0A04585B5B5B5B0000EA
S1131410000000000000000B010C9BD1857BD0742
S113142036C1012708CE0216BD0DFB20397D02B559
S11314302743BD0897C1FF2608CE0211BD0DFB202E
S113144025FF0CA8FE02C2A6088A04A708BD1478CA
S1131450F60CA8BD186BF60CA9BD186BC652BD18C6
S11314607FC64EBU187F730CA67302BA7C0CA57C94
S11314700CA5BD0C4E7E04D0FE02C28606E60036E4
S1131480FF148FBD186B32FE148F084A26EF390003
S113149000BD10C9FE02C2FF14EA7D02B62608CE02
S11314A002138D0DFB203BD00736C10D2605CE023E
S11314B01620EFBD009F5C1FF27E87D02B5261CFE05
S11314C014EAA6037D02B7260434BF27028A40A736
S11314D008B60CA9A707B60CA8A706730CA67C0C23
S11314E0A57C0CA5B00C4E7E04D00000BD10C9BD6A
S11314F01657BD0736C1012708CE0216BD0DFB2023
S11315000C4E20477C02C47C02C47C02C47D02B51C
S1131510260EBD0838FE02C2A6088A08A70820289D
S1131520C67EF70CA7BD186BBD0897BD1478C653C6
S1131530BD187F70CA87F0CA97C0CA57C0CA57C16
S11315400CA57302B9BD0C4EBU12027E04D0BD1081
S1131550CYBD0736C10D2608CE0216BD0DFB201AE3
S1131560BD09F5BD11177C02C47D02B5270FF60C29
S1131570A9BD186B7C0CA57C0CA6BD0C4EBU12023B
S11315807E04D0BD10C9BD0736C1272708CE02048A
S1131590BD0DFB203CFE02BEE600C10D27EFF70C9B
S11315A0A97D02B52703BD186BF02BEE0BF02BEE6B
S11315B07C02C4E600C1272706C10D26E4200CE600
S11315C001C127260608F02BE20D67C0CA67C0C3F
S11315D0A5BD0C4EBU12027E04D0BD10C9BD073698
S11315E0C10D2608CE0216BD0DFB2039BD09F5BD7F
S11315F011177C02C47C02C47D02B5272B7F0CA80B
S1131600BD186BF60CA9BD186B7D02B72704C65232
S113161020077D02B82705C64BD187F7C0CA57C2C
S11316200CA5730CA6BD0C4EBU12027E04D0BD10D9
S1131630C9BD1857BD0736C10D2608CE0216BD0D0B
S1131640FB2023BD09F5C1FF27F1BD17A77D03B713
S113165027147D0CA8260A7D0CA926057F03B72034
S11316600586FB703B7BD0C4E7E04D00000BD1045

S1131670C9BD0C4E7F166C7F166D7D02B526307D7C
S113168002B6260B73166DCE0226BD0DFB2020FE7E
S113169002C28620A708B6034DA706B6034EA707C5
S11316A0BD0736FE02BBA6008143260373166C8D3C
S11316B005B6FE02A708FF02AFBD0C4EFE02C0A687
S11316C000812A260A7D166C27E5BD172320E07FBA
S11316D002B6FE02C0A60081202706BD07367302AB
S11316E0B6BD07368604B70827FE02B8FF0823CE23
S11316F0175FF0825CE0823BD0705260U7D02B61A
S1131700270CE0227BD0DFB2006BD17237E16AF84
S11317107D02B526088617FE034DA70008FF034D77
S11317207E04D07D02B526367D166D2631FE02C08C
S1131730FF02B8FE02BEA60008FF02B8FE034D8CB1
S113174002A42610860DA70008FF034DCE0228BD73
S11317500DFB200AA70008FF034D810D26D5394D46
S1131760454E44BD10C9BD1857BD0736C1012709F0
S1131770CE0216BD0DFB7E14667D02B52606BD089D
S1131780387E142DF60353BD186BF60354BD186B45
S1131790C650BD187F7E142BD10C9BD1857BD1736
S11317A0BBBD0C4E7E04D0BF02CCFE03B58C03AD92
S11317B0271DBE03B5B603B736201BBF02CCFE03FC
S11317C0B58C03B52709BE03B532B703B72007CEDE
S11317D00254BD0DFB39B03B5BE02CC39BD10C9DF
S11317E0BD18577D02B52713F602C7270EC63CF075
S11317F002C7BD13FE5A26FA7F02C77E04D0BD106D
S1131800C9BD0736C10D2608CE0216BD0DFB201832
S1131810BD09F5C1FF27F47D02B5270FBD18407C33
S11318200CA57C0CA5730CA6BD0C4EBU1202B4F60236
S1131830B3B80CA9F90CA8B702B4F702B37E04D069
S11318405FFE0CA8FF02C5BD186BF02C509270682
S1131850FF02C55F20F1397D02B6270E7D02B52651
S113186003BD08EBCE0223BD0DFB39B602AE8540A5
S1131870260C17BD16BD0296178D14BD029639B627
S113188002AE854026F817BD029639444444444488
S11318900F8B30813923028B0739BD18C508A60088
S11318A0810426F639A6008D0EA60008200D8DF5BC
S11318B08DF386207E18C544444444440F8B3081C4
S11318C03923028B0736BD03C832810A260E363705
S11318D0C6088600BD03CB5A26F83332397EE83871
S11318E07EE8207EE800000037FF18E6BDE929FE07
S11318F018E633937FF18E6BDE9AAFE18E633398E
S10E19004FBDE9AA910D26F839190B20
59

APPENDIX H

ASCII Text Listing of the Relocatable Format Object Code for RA6800ML

The listing below gives the relocatable format object code of the relocatable macro assembler RA6800ML in ASCII text form. This listing can be used to either enter the program by hand or to verify the entry of the program via the bar codes given in Appendix I. Note that the ends of lines in this verification listing *do not* represent line feed or carriage return codes within the machine readable text.

The relocatable file of the macro assembler can be run through the relocatable linking loader LINK68, available as the PAPERBYTE™ publication *LINK68: Linking Loader for the Motorola M6800* by Robert Grappel and Jack Hemenway (ISBN 0-931718-09-0), in order to reposition RA6800ML at an arbitrary, more convenient address if low memory is not the ideal location in the user's system. This form of the Assembler object code will not be needed by users who can employ the absolute object code version of RA6800ML given in Appendices D or E without further relocation.

Appendix G gives an assembly language source listing for RA6800ML.

[illegible]

143

ERFE13D6RC6FFE700BD13BER7E128DRBD13BERBD13BERCE13A1RB60288RF6028
9RBD0D5CRCE139SRBD185ERBD13BERBD13BERCE13AERBD185ERCE0313RBD186E
RBD13BERB6026ER85402606BD0259R7E024DR7E0250R205245444546494E4544
04544845524520574552453A200000000000204552524F525304434F4D4D4F4E
204C454E4754483D2004CE13C5RBD185ER390D0A045B5B5B5B5B5B0000000000
0000000000BD1089RBD1817RBD06F6RC1012708CE0216BD0DBBR20397D0275R2
743BD0857RC1FF2608CE0211BD0DBBR2025FF0C68RFE0282RA6088A04A708BD1
438RF60C68RBD182BRF60C69RBD182BRC652BD183FRC64EBD183FR730C66R730
27AR7C0C65R7C0C65RBD0C0ER7E0490RFE0282R8606E60036FF144FRBD182BR3
2FE144FR084A26EF390000BD1089RFE0282RFF14AAR7D0276R2608CE0213BD0D
BBR203DBD06F6RC10D2605CE021620EFBD09B5RC1FF27E87D0275R261CFE14AA
RA6087D0277R260484BF20028A40A708B60C69RA707B60C68RA706730C66R7C0
C65R7C0C65RBD0C0ER7E0490R0000BD1089RBD1817RBD06F6RC101270BCE0216
BD0DBBRBD0C0ER20477C0284R7C0284R7C0284R7D0275R260EBD07F8RFE0282R
A6088A08A7082028C67EF70C67RBD182BRBD0857RBD1438RC658BD183FR7F0C6
8R7F0C69R7C0C65R7C0C65R7C0C65R730279RBD0C0ERBD11C2R7E0490RBD1089
RBD06F6RC10D2608CE0216BD0DBBR201ABD09B5RBD10D7R7C0284R7D0275R270
FF60C69RBD182BR7C0C65R7C0C66RBD0C0ERBD11C2R7E0490RBD1089RBD06F6R
C1272708CE0204BD0DBBR203CFE027ERE600C10D27EFF70C69R7D0275R2703BD
182BRFE027ER08FF027ER7C0284RE600C1272706C10D26E4200CE601C1272606
08FF027ER20D67C0C66R7C0C65RBD0C0ERBD11C2R7E0490RBD1089RBD06F6RC1
0D2608CE0216BD0DBBR2039BD09B5RBD10D7R7C0284R7C0284R7D0275R272BF6
0C68RBD182BRF60C69RBD182BR7D0277R2704C65220077D0278R2705C64DBD18
3FR7C0C65R7C0C65R730C66RBD0C0ERBD11C2R7E0490RBD1089RBD1817RBD06F
6RC10D2608CE0216BD0DBBR2023BD09B5RC1FF27F1BD1767R7D0377R27147D0C
68R260A7D0C69R26057F0377R200586FFB70377RBD0C0ER7E0490R0000BD1089
RBD0C0ER7F162CR7F162DR7D0275R26307D0276R260B73162DRCE0226BD0DBBR
2020FE0282R8620A708B6030DRA706B6030ERA707BD06F6RFE027BRA60081432
60373162CRBD0576RFE026FR08FF026FRBD0C0ERFE0280RA600812A260A7D162
CR27E5BD16E3R20E07F0276RFE0280RA60081202706BD06F6R730276RBD06F6R
8604B707E7RFE027BRFF07E3RCE171FRFF07E5RCE07E3RBD06C5R260D7D0276R
270ECE0227BD0DBBR2006BD16E3R7E166FR7D0275R260B8617FE030DRA70008F
F030DR7E0490R7D0275R26367D162DR2631FE0280RFF027ERFE027ERA60008FF
027ERFE030DRBC0264R2610860DA70008FF030DRCE0228BD0DBBR200AA70008F
F030DR810D26D5394D454E44BD1089RBD1817RBD06F6RC1012709CE0216BD0DB
BR7E1426R7D0275R2606BD07F8R7E13EDRF60313RBD182BRF60314RBD182BRC6
50BD183FR7E13EDRBD1089RBD1817RBD177BRBD0C0ER7E0490RBF028CRFE0375
R8C036DR271DBE0375RB60377R36201BBF028CRFE0375R8C0375R2709BE0375R
32B70377R2007CE0254BD0DBBR39BF0375RBE028CR39BD1089RBD1817R7D0275
R2713F60287R270EC63CF00287RBD13BER5A26FA7F0287R7E0490RBD1089RBD0
6F6RC10D2608CE0216BD0DBBR2018BD09B5RC1FF27F47D0275R270FBD1800R7C
0C65R7C0C65R730C66RBD0C0ERB60274RF60273RBB0C69RF90C68RB70274RF70
273R7E0490R5FFE0C68RFF0285RBD182BRFE0285R092706FF0285R5F20F1397D
0276R270E7D0275R2603BD08ABRCE0223BD0DBBR39B6026ER8540260C178D16B
D0256R178D14BD0256R39B6026ER854026F817BD0256R39444444444840F8B308
13923028B0739BD1885R08A600810426F639A6008D0EA60008200D8DF58DF386
207E1885R444444444840F8B30813923028B0736BD038BR32810A260E3637C608
8600BD038BR5A26F8333239

APPENDIX I

PAPERBYTE™ Bar Code Representation of Relocatable Format Object Code for RA6800ML

Beginning on the following page is a complete machine readable representation (PAPERBYTETM bar codes) of the relocatable object code for the relocatable macro assembler RA6800ML. The format is that of an ASCII text string without carriage return or line feed conventions. Appendix H is a direct listing of this file using fixed length lines to make it fit the confines of a printed page.

This representation uses the bar code text format, in which each bar code frame (one line of bar codes running from top to bottom of the page) contains a segment of the ASCII relocatable format object text. The text must be loaded into memory and then saved on the user's mass storage device. For details on the text format used in this and other PAPERBYTETM books, see the PAPERBYTE publication *Bar Code Loader* by Ken Budnick. The book contains a brief history on bar codes, a general bar code loader algorithm with flowcharts, and complete program listing for 6800, 6502, and 8080 or Z-80 based systems.

The relocatable file of the macro assembler can be run through the relocatable linking loader LINK68, available as the PAPERBYTETM publication *LINK68: Linking Loader for the Motorola M6800* by Robert Grappel and Jack Hemenway (ISBN 0-931718-09-0), in order to reposition RA6800ML at an arbitrary, more convenient address if low memory is not the ideal location in the user's system. This form of the Assembler object code will not be needed by users who can employ the absolute object code version of RA6800ML given in Appendices D or E without further relocation.

Appendix G gives an assembly language source listing for RA6800ML.

[illegible]

[illegible]

[illegible]

4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4
0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	2	2	2
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9

1
 2
 3
 4
 5
 6
 7
 8
 9
 10
 11
 12
 13
 14
 15
 16
 17
 18
 19
 20
 21
 22
 23
 24
 25
 26
 27
 28
 29
 30
 31
 32
 33
 34
 35
 36
 37
 38
 39
 40
 41
 42
 43
 44
 45
 46
 47
 48
 49
 50
 51
 52
 53
 54
 55
 56
 57
 58
 59
 60
 61
 62
 63
 64
 65
 66
 67
 68
 69
 70
 71
 72
 73
 74
 75
 76
 77
 78
 79
 80
 81
 82
 83
 84
 85
 86
 87
 88
 89
 90
 91
 92
 93
 94
 95
 96
 97
 98
 99
 100
 101
 102
 103
 104
 105
 106
 107
 108
 109
 110
 111
 112
 113
 114
 115
 116
 117
 118
 119
 120
 121
 122
 123
 124
 125
 126
 127
 128
 129
 130
 131
 132
 133
 134
 135
 136
 137
 138
 139
 140
 141
 142
 143
 144
 145
 146
 147
 148
 149
 150
 151
 152
 153
 154
 155
 156
 157
 158
 159
 160
 161
 162
 163
 164
 165
 166
 167
 168
 169
 170
 171
 172
 173
 174
 175
 176
 177
 178
 179
 180
 181
 182
 183
 184
 185
 186
 187
 188
 189
 190
 191
 192
 193
 194
 195
 196
 197
 198
 199
 200
 201
 202
 203
 204
 205
 206
 207
 208
 209
 210
 211
 212
 213
 214
 215
 216
 217
 218
 219
 220
 221
 222
 223
 224
 225
 226
 227
 228
 229
 230
 231
 232
 233
 234
 235
 236
 237
 238
 239
 240
 241
 242
 243
 244
 245
 246
 247
 248
 249
 250
 251
 252
 253
 254
 255
 256
 257
 258
 259
 260
 261
 262
 263
 264
 265
 266
 267
 268
 269
 270
 271
 272
 273
 274
 275
 276
 277
 278
 279
 280
 281
 282
 283
 284
 285
 286
 287
 288
 289
 290
 291
 292
 293
 294
 295
 296
 297
 298
 299
 300
 301
 302
 303
 304
 305
 306
 307
 308
 309
 310
 311
 312
 313
 314
 315
 316
 317
 318
 319
 320
 321
 322
 323
 324
 325
 326
 327
 328
 329
 330
 331
 332
 333
 334
 335
 336
 337
 338
 339
 340
 341
 342
 343
 344
 345
 346
 347
 348
 349
 350
 351
 352
 353
 354
 355
 356
 357
 358
 359
 360
 361
 362
 363
 364
 365
 366
 367
 368
 369
 370
 371
 372
 373
 374
 375
 376
 377
 378
 379
 380
 381
 382
 383
 384
 385
 386
 387
 388
 389
 390
 391
 392
 393
 394
 395
 396
 397
 398
 399
 400
 401
 402
 403
 404
 405
 406
 407
 408
 409
 410
 411
 412
 413
 414
 415
 416
 417
 418
 419
 420
 421
 422
 423
 424
 425
 426
 427
 428
 429
 430
 431
 432
 433
 434
 435
 436
 437
 438
 439
 440
 441
 442
 443
 444
 445
 446
 447
 448
 449
 450
 451
 452
 453
 454
 455
 456
 457
 458
 459
 460
 461
 462
 463
 464
 465
 466
 467
 468
 469
 470
 471
 472
 473
 474
 475
 476
 477
 478
 479
 480
 481
 482
 483
 484
 485
 486
 487
 488
 489
 490
 491
 492
 493
 494
 495
 496
 497
 498
 499
 500
 501
 502
 503
 504
 505
 506
 507
 508
 509
 510
 511
 512
 513
 514
 515
 516
 517
 518
 519
 520
 521
 522
 523
 524
 525

4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
8	8	8	8	8	8	8	8	8	9	9	9	9	9	9	9	9	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1

[illegible]

5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	
2	2	2	2	2	2	2	2	2	2	3	3	3	3	3	3	3	3	3	4	4	4	4	4	4	4	4	4	4	4	5	5	5	5	5	5	5	5	5	
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9

[illegible]

6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6
0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	2	2	2	2
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9

APPENDIX J

Cassette Tape IO Listing


```

0000 0000 N      NAM TDRIVERS
*
*      TAPE DRIVERS FOR RA6800ML ASSEMBLER
*      COPYRIGHT 1977 JACK E. HEMENWAY
*      BOSTON MASS. 02111 ALL RIGHTS RESERVED
*
*      ROUTINES IN THE ASSEMBLER
*
0000 7E 0000 X      EXT PDATA1
0003 7E 0000 X      EXT INEEF
0006 7E 0000 X      EXT CRLF
*
*      ENTRY POINTS IN DRIVER
*
0009 01E1 N      ENT TABLES
0009 0009 N      ENT UPDATE
0009 000C N      ENT MONITOR
0009 0016 N      ENT GETB
0009 0032 N      ENT OUTB
0009 0088 N      ENT WREOF
0009 004E N      ENT INITIO
0009 005F N      ENT RESTR
*
*      LOCATIONS IN MIKBUG
*
0009 7E E0E3 UPDATE JMP $E0E3
000C 7E E0E3 MONITOR JMP $E0E3
*
000F 0001 CKSUM RMB 1
0010 0002 INPTR RMB 2
0012 0002 OIPTR RMB 2
0014 0002 JXSV RMB 2
*
*      GET A BYTE RETURN IN A REGISTER
*
0016 FF 0014 R GETB STX DXSV
0019 FE 0010 R      LDX INPTR
001C A6 00          LDA A 0,X      GET A CHAR
001E 81 17          CMP A #$17     ETB ?
0020 26 05          BNE GETBI      NO
*
0022 37            PSH B
0023 BD 009A R      JSR RDBUF      READ ANOTHER BLOCK
0026 33            PUL B
*
0027 A6 00 GETBI LDA A 0,X      GET CHAR
0029 08            INX
002A FF 0010 R      STX INPTR
002D FE 0014 R      LDX DXSV
0030 0C            CLC
0031 39            RTS
*
*
*      OUTPUT BYTE IN A REGISTER
*
0032 FF 0014 R OUTB STX DXSV
0035 FE 0012 R      LDX OIPTR
0038 8C 05E0 R      CPX #OIBUF+$1FD FULL?
003B 26 07          BNE OUTBI      NO
*
003D 36            PSH A
003E 37            PSH B
003F BD 0129 R      JSR WRITBF
0042 32            PUL A
0043 33            PUL B
*
0044 A7 00 OUTBI STA A 0,X      SAVE CHAR
0046 08            INX
0047 FF 0012 R      STX OIPTR
004A FE 0014 R      LDX DXSV
004D 39            RTS
*
*
004E CE 01E3 R INITIO LDX #INBUF
0051 FF 0010 R      STX INPTR
0054 86 17          LDA A #$17
0056 A7 00          STA A 0,X
*
0058 CE 03E3 R      LDX #OTBUF
005B FF 0012 R      STX OIPTR
005E 39            RTS
*
*      PROMPT TO REWIND TAPE

```

```

*
005F CE 0070 R RESTR LDX #REWIND
0062 BD 0000 R JSR PDATA1
0065 BD 0003 R JSR INEE
0068 81 00 CMP A #SOD CR ?
006A 26 F9 BNE *-5

*
006C BD 0006 R JSR CRLF
006F 39 RTS

*
0070 52 REWIND FCC 'REWIND TAPE AND TYPE CR'
0077 04 FCB 4

*
* CLOSE OUTPUT FILE
*
0088 BD 0129 R #REOF JSR WRITBF
008B FE 0012 R LDX OIPTR
008E 86 04 LDA A #4
0090 A7 00 STA A 0,X
0092 08 INX
0093 FF 0012 R STX OIPTR
0096 BD 0129 R JSR WRITBF
0099 39 RTS

* READ IN A BLOCK FROM TAPE 1 *
*
009A 7F 000F R RDBUF CLR CKSUM
009D CE 01E3 R LDX #INBUF POINT TO INBUF
00A0 BD 015E R JSR TIINZ START TAPE 1
00A3 BD 0181 R RD1 JSR TIGET GET CHAR
00A6 5D TST B OK ?
00A7 26 18 BNE RD2 NO

*
00A9 A7 00 STA A 0,X PUT IN INBUF
00AB 03 INX BUMP POINTER
00AC 81 04 CMP A #S04 EOF?
00AE 27 1D BEQ RD4 YES
00B0 81 17 CMP A #S17 ETB?
00B2 26 EF BNE RD1 NO
00B4 8C 03E2 R CPX #INBUF+S1FF OVERRUN ?
00B7 27 08 BEQ RD2 YES
00B9 BD 0181 R JSR TIGET GET CKSUM BYTE
00BC 7C 000F R INC CKSUM OK ?
00BF 27 05 BEQ RD3 YES

*
00C1 CE 0102 R RD2 LDX #TAPERR BAD
00C4 2J 0A BRA RD5 RD5 FINISH UP

*
00C6 BD 0199 R RD3 JSR TIISTP STOP TAPE 1
00C9 CE 01E3 R LDX #INBUF INIT INPTR
00CC 39 RTS

*
00CD CE 00E0 R RD4 LDX #EOF EOF MSG
00D0 BD 0199 R RD5 JSR TIISTP STOP TAPE
00D3 BD 0000 R JSR PDATA1 PRINT MESSAGE
00D6 BD 0003 R JSR INEE WAIT FOR "GO"
00D9 81 00 CMP A #SOD CR ?
00DB 26 F9 BNE *-5 NO
00DD 7E 009A R JMP RDBUF TRY AGAIN

*
00E0 45 EOF FCC 'EOF*REPOSITION TAPE AND TYPE CR'
00FF 000A FDB $0DOA CR,LF
0101 04 FCB 4 EOT

*
0102 54 TAPERR FCC 'TAPE ERROR*BACK UP A BLOCK & TYPE CR'
0126 0DOA FDB $0DOA CR,LF
0128 04 FCB $04 EOT

* WRITBF: WRITE OUT OIBUF TO TAPE2
*
0129 37 WRITBF PSH B
012A FE 0012 R LDX OIPTR
012D 8C 03E3 R CPX #OIBUF EMPTY
0130 27 22 BEQ WRITBFC YES

*
0132 86 17 LDA A #S17 LOAD ETB
0134 A7 00 STA A 0,X PUT INTO OIBUF
0136 CE 03E3 R LDX #OIBUF POINT TO OIBUF
0139 5F CLR B CLR CKSUM REG
013A BD 01A1 R JSR T20TZ START TAPE

*
013D A6 00 WRITBFA LDA A 0,X GET CHAR
013F EB 00 ADD B 0,X ADD TO CKSUM
0141 BD 01BC R JSR T20UT
0144 8C 0012 R CPX OIPTR DONE ?
0147 27 03 BEQ WRITBFB

*
0149 08 INX NO
014A 20 F1 BRA WRITBFA DO AGAIN

*
014C 53 WRITBFB COM B FORM CKSUM
014D 17 TBA BYTE
014E BD 01BC R JSR T20UT
0151 BD 01C9 R JSR T20STP STOP TAPE

```

```

0154 CE 03E3 R WRTBFC LDX #OTBUF
0157 FF 0012 R STX OTPTR INIT OTPTR
015A 33 PUL B
015B 39 RTS

```

* TAPE DRIVERS:

```

015C 8010 TPIST EQU $8010
015C 8011 TPIDAT EQU $8011
015C 8014 TP2ST EQU $8014
015C 8015 TP2DAT EQU $8015
015C 0002 TXSV RMB 2

```

* START TAPE FOR A READ:

```

015E FF 015C R FIINZ STX TXSV
0161 36 PSH A
0162 86 17 LDA A #$17 MASTER RESET, RTS:=0
0164 B7 8010 STA A TPIST

```

```

0167 86 5D LDA A #$5D RTS:=1
0169 B7 8010 STA A TPIST

```

```

016C CE 0280 LDX #0280 DELAY 1 SEC
016F BD 01D9 R JSR IDELY

```

```

0172 86 57 LDA A #$57 MASTER RESET
0174 B7 8010 STA A TPIST
0177 86 5D LDA A #$5D RTS:=1
0179 B7 8010 STA A TPIST
017C 32 PUL A
017D FE 015C R LDX TXSV
0180 39 RTS

```

* READ A BYTE

```

0181 F6 8010 FIIGET LDA B TPIST GET STATUS
0184 C5 01 BIT B #$01 RDRF?
0186 27 F9 BEQ *-5 NO

```

```

0188 C5 70 BIT B #$70 ERRORS?
018A 27 01 BEQ *-3 NO

```

```

018C 39 RTS YES

```

```

018D B6 8011 LDA A TPIDAT GET BYTE
0190 16 TAB
0191 FB 000F R ADD B CKSUM FORM CHECKSUM
0194 F7 000F R STA B CKSUM
0197 5F CLR B
0198 39 RTS

```

* STOP TAPE AFTER A READ

```

0199 36 FIISTP PSH A
019A 86 17 LDA A #$17
019C B7 8010 STA A TPIST
019F 32 PUL A
01A0 39 RTS

```

* START TAPE FOR OUTPUT

```

01A1 37 T20TZ PSH B
01A2 36 PSH A
01A3 FF 015C R STX TXSV
01A6 C6 17 LDA B #$17 MASTER RESET
01A8 F7 8014 STA B TP2ST
01AB C6 5D LDA B #$5D RTS:=1
01AD F7 8014 STA B TP2ST

```

```

01B0 CE 0500 LUX #0500 DELAY 2 SECS.
01B3 BD 01D9 R JSR IDELY

```

```

01B6 32 PUL A
01B7 33 PUL B
01B8 FE 015C R LUX TXSV
01BB 39 RTS

```

* WRITE A BYTE TO TAPE

```

01BC 37 T20UT PSH B
01BD F6 8014 T20UTA LDA B TP2ST GET STATUS
01C0 C5 02 BIT B #$02 RDRF?
01C2 27 F9 BEQ T20UTA NO

```

```

01C4 B7 8015 STA A TP2DAT YES, WRITE BYTE
01C7 33 PUL B
01C8 39 RTS

```

* STOP TAPE AFTER A WRITE

```

01C9 4F T20STP CLR A WRITE PAD CHARS
01CA BD 01BC R JSR T20UT
01CD BD 01BC R JSR T20UT
01D0 BD 01BC R LDA A #$17
01D3 86 17 STA A TP2ST
01D5 B7 8014 RTS
01D8 39

```

```

01D9 4F IDELY CLR A
01DA 4C IDELY1 INC A
01DB 26 FD BNE IDELY1

```

```

01DD 09 DEX
01DE 26 FA BNE IDELY1
01E0 39 RTS

```

```

01E1 05E4 R TABLES FCB **$0403
01E3 01E3 R INBUF EQU *
01E3 03E3 R OTBUF EQU **$200

```

END

```

CKSUM 000F R
CRLF 0006 RX
DXSV 0014 R
EOF 00E0 R
GETB 0016 RN
GETBI 0027 R
INBUF 01E3 R
INEEE 0003 RX
INITIO 004E RN
INPTR 0010 R
MONIOR 000C RN
OTBUF 03E3 R
OTPTR 0012 R
OUTB 0032 RN
OUTBI 0044 R
PDATA1 0000 RX
RD1 00A3 R
RD2 00C1 R
RD3 00C6 R
RD4 00CD R
RD5 00D0 R
RDBUF 009A R
RESTR 005F RN
REWIND 0070 R
FIIGET 0181 R
FIINZ 015E R
FIISTP 0199 R
T20STP 01C9 R
T20TZ 01A1 R
T20UT 01BC R
T20UTA 01BD R
TABLES 01E1 RN
TAPERH 0102 R
IDELY 01D9 R
IDELY1 01DA R
TURIVE 0000 RN
TPIDAT 8011
TPIST 8010
TP2DAT 8015
TP2ST 8014
TXSV 015C R
UPDATE 0009 RN
WREOF 0088 RN
WRTBFC 0154 R
WRTBFA 013D R
WRTBFB 014C R

```

THERE WERE: 00000 ERRORS

COMMON LENGTH= 0000

ICOM FD0S-II/6800-0.1

!

Appendix K

ICOM Floppy Disk IO Listing

[illegible]

Index

ABRCK 39, 41-43, **48**, 2585
 Accumulator Addressing Mode 2-4, 39, 42, 43, 49
 ADDR1 14, **39**, 46, 48-52, 56, 2725
 ADDR2 14, **41**, 48-52, 56, 2217
 ADDR3 14, **42**, 48-52, 56, 2287
 ADDR4 14, **43**, 48, 49, 52, 2337
 ADDR5 14, **44**, 48-52, 56, 2366
 ADDR6 **45**, 48, 52, 2438
 ADDR7 14, **46**, 48-52, 56, 2458
 ADDR8 14, **47-50**, 52, 2493
 ADDR9 14, 19, **48**, 2544
 ADD16 14, 16, 18, 55, 56, 1807
 ADRINT 11, 24-32, 34-37, 39, 41-**48**, 2565
 Cassette Tape Files 19, 20-23, **60**, 67
 Character Set **1**
 Character Table (CHRTAB) **12**, 13, 0279
 CMN **5**, 7
 Comments **1**, 2
 Common 5, 7, 24, 56
 COMPAR **13**, 16, 19, 25, 34, 0977
 CRLF 25, 3077
 CVBTD **13**, 25, 2078
 CVDB **13**, 56, 1677
 CVHB **13**, 56, 1617
 CVHBS **13**, 1660
 DELSYM **19**, 38, 1284
 Direct Addressing Mode **3**, 4, 39, 41, 44, 45, 50
 Diskette Files 19-23, **61**
 DIV16 **14**, 18, 56, 1749
 DSCAN **55**, 1047
 END **5**, 11, 12, 25
 ENT **5**, 7, 26
 Entry **7**
 EQU **5**, 7, 27
 EXT **5**, 7, 28
 Extended **3**, 4, 39, 41, 42, 44-46, 51
 External **7**
 FCB **5**, 29
 FCC **5**
 FDB **5**, 7, 31
 GETB 23, 0350

GCHRTB 14, 52, 55, 1100
 HASH 14, 17, 18, 1298
 HSCAN 52, 55, 1077
 IF 5, 13, 32
 Immediate Addressing Mode 3, 4, 39, 44, 50, 51
 Indexed Addressing Mode 2-4, 39, 41, 42, 44-46, 49, 50, 55, 56
 INEEE 23, 0408
 Inherent Addressing Mode 2, 5, 39, 48
 INXCK 14, 39, 41, 42, 44, 56, 49, 52, 2598
 Label 1, 5, 6, 38
 LBLCK 14, 19, 24-26, 28, 32, 35, 36, 38, 3622
 LCLCN 28-31, 39, 41-44, 46-48, 52, 2759
 LCNAB1 14, 19, 42, 43, 49, 2634
 LCNAB2 39, 50, 2658
 LCNAB3 39, 41, 51, 2702
 LCN2 14, 19, 42, 44, 46, 47, 50, 2678
 LCN3 14, 19, 42, 44, 46, 51, 2721
 LINCK 14, 25, 1851
 LKPSYM 11, 18, 19, 24, 26, 28, 56, 1216
 MAC 6, 34
 MACMOV 14, 35, 3445
 MACPSH 11, 22, 0806
 MACPUL 22, 0870
 Macro 2, 6, 7, 8, 20-22, 25, 34, 35
 Macro Table (MACTBL) 6, 11, 12, 13, 0360
 MAIN1 0541
 MEND 6-8, 34
 Mnemonic Table (MNTAB) 11, 12, 16, 0018
 MNLKP 11, 13, 14, 16, 1384
 MPY16 13, 14, 16, 18, 56, 1718
 NAM 6, 35
 NIF 6, 13, 36
 NSCAN 55, 1056
 NSEVL 13, 14, 18, 24, 27, 29, 31, 32, 37, 41, 42, 44, 46, 47, 49, 56, 1457
 NXTOK 11, 24, 26-32, 34, 35, 37, 41-47, 49, 52, 55, 56, 0976
 Opcode 2, 39
 Operand 2, 39
 OUTB 19, 0351
 OUTBIN 19, 26, 28-31, 35, 38, 48-50, 3638
 OUTBNR 19, 26, 28, 31, 35, 3651
 OUTCHR 14, 25, 36, 3703

OUTHL 19, 3660
 OUTHR 19, 3664
 OUTL 14, 1891
 OUTL7A 14, 1988
 OUT2HS 14, 3687
 OUT4HS 14, 3686
 PAG 6, 36
 PASS1 0417
 PASS2 0522
 PBLOCK 28, 3085
 PDATA1 14, 16, 23, 25, 3675
 POCMN 14, 17, 18, 38, 48, 52, 2769
 POEND 13, 14, 25, 38, 48, 2832
 POENT 14, 18, 19, 26, 28, 38, 48, 52, 3037
 POEQU 14, 27, 48, 52, 56, 3104
 POEXT 14, 17-19, 28, 38, 48, 52, 3155
 POFCB 14, 19, 29, 48, 49, 52, 56, 3201
 POFCC 14, 19, 30, 48, 52, 3227
 POFDB 14, 19, 31, 48, 52, 56, 3273
 POIF 14, 32, 33, 48, 56, 3316
 POMAC 13, 14, 20, 34, 35, 48, 52, 3354
 PONAM 14, 17, 19, 35, 38, 48, 52, 3483
 PONIF 14, 33, 36, 48, 3512
 POPAG 36, 38, 48, 3553
 PORMB 14, 37, 38, 48, 52, 56, 3572
 PRINTE 11, 13, 14, 17, 22, 24-32, 34-39, 41-44, 46, 47, 49, 2072
 PRINTL 11, 14, 24-32, 34, 35, 37, 48-51, 1833
 PSHIF 33, 3521
 PULIF 33, 36, 3521
 P2ERR 14, 31, 39, 41, 42, 44, 46, 47, 49, 2612
 RDBUF 23
 RDLINE 11, 20, 21, 34, 0668
 RDMAC 14, 20, 21, 22, 0711
 Relative Addressing Mode 3, 4, 39, 47
 RESTR 25
 RMB 6, 37
 RMBOUT 19, 37, 38, 3603
 SPACER 14, 16, 1866
 Statement Characteristics 1
 Statement Length 1

STOSYM 11, 14, 17-19, 24, 28, *1141*
SUB16 14, 56, *1819*
SYMCMP 13, 17-19, *1250*
Symbol Table (SYMTAB) 11, 12, 13, 16-19, 24, 26-28, 34, 38, *0362*
SYMMOD 17, 18, *1263*
TDELY 59
TIGET 23, 59
T1INZ 23, 59
T1ISTP 23, 59
T20STP 59
T20TZ 59
T2OUT 59
WREOF 25, 59, *0352*
WRITBF 59

NOTE: The page numbers in bold type face indicate either the definition or the primary reference to the item. The numbers in *italics* indicate the line number in the source code listing.

BYTE Publications, Inc.
Production Credits

Blaise Liffick — Technical Editor
Edmond Kelly Jr. — Production Manager
Patricia Curran — Production Assistant
Walter Banks, University of Waterloo, CCNG, Bar Codes
George Banta Company, Printing
Dawson Advertising, Cover Art

A Note About Bar Codes . . .

Bar codes are the newest form of machine readable data representation. They are used in all PAPERBYTE™ software products in BYTE magazine articles and self contained book publications and combine efficiency of space, low cost, and ease of data entry with the need for mass produced machine readable representations of software. Bar codes were originally used for product identification in inventory control and supermarket checkout applications. Today, because of their direct binary representation of data, they are an ideal computer compatible communications medium. In the application of bar codes to software distribution (such as PAPERBYTE books and articles), the use of a simple but reliable optical scanning wand and an appropriate program provides a convenient means for the user to acquire software.

Our intent in making PAPERBYTE software available in bar code form is to provide a method of conveying machine readable information from documentation to the memories and mass storage of a user's system on a one time basis. We suggest that the user of software obtained in this manner should locally record the data on the mass storage devices of his system after the data has been scanned from the printed page. The PAPERBYTE bar code representations provide a standardized means of obtaining the data, but they cannot be compared to the convenience of local mass storage devices such as floppy disks, digital cassettes or audio cassettes. Thus if repeated use of the software obtained from bar code is anticipated, we recommend that the user make a copy on some form of magnetic medium.

Bar Code Loader by Ken Budnik, the first in the PAPERBYTE series of software books, provides a brief history of bar codes, a look at the PAPERBYTE bar code format including flowcharts, a general bar code loader algorithm and well documented programs with complete implementation and checkout procedures for 6800, 6502 and 8080/Z-80 based systems.

RA6800ML:

AN M6800 RELOCATABLE MACRO ASSEMBLER is a two pass assembler for the Motorola 6800 microprocessor. It is designed to run on a minimum system of 16 K bytes of memory, a system console (such as a Teletype terminal), a system monitor (for instance, the Motorola MIKBUG read only memory program or the ICOM Floppy Disk Operating System), and some form of mass file storage (dual cassette recorders or a floppy disk).

The Assembler can produce a program listing, a sorted Symbol Table listing, and relocatable object code. The object code is loaded and linked with other modules using the Linking Loader **LINK 68**. (Refer to **PAPERBYTE™** publication **LINK 68: AN M6800 LINKING LOADER** for details).

Included in this book: a complete description of the 6800 assembly language and its components, including outlines of the instruction and address formats, pseudo instructions, and macro facilities; details on interfacing and using the Assembler; error messages generated by the Assembler; the Assembler and sample IO driver source code listings; and the **PAPERBYTE™** bar code representation of the Assembler's relocatable object file.

